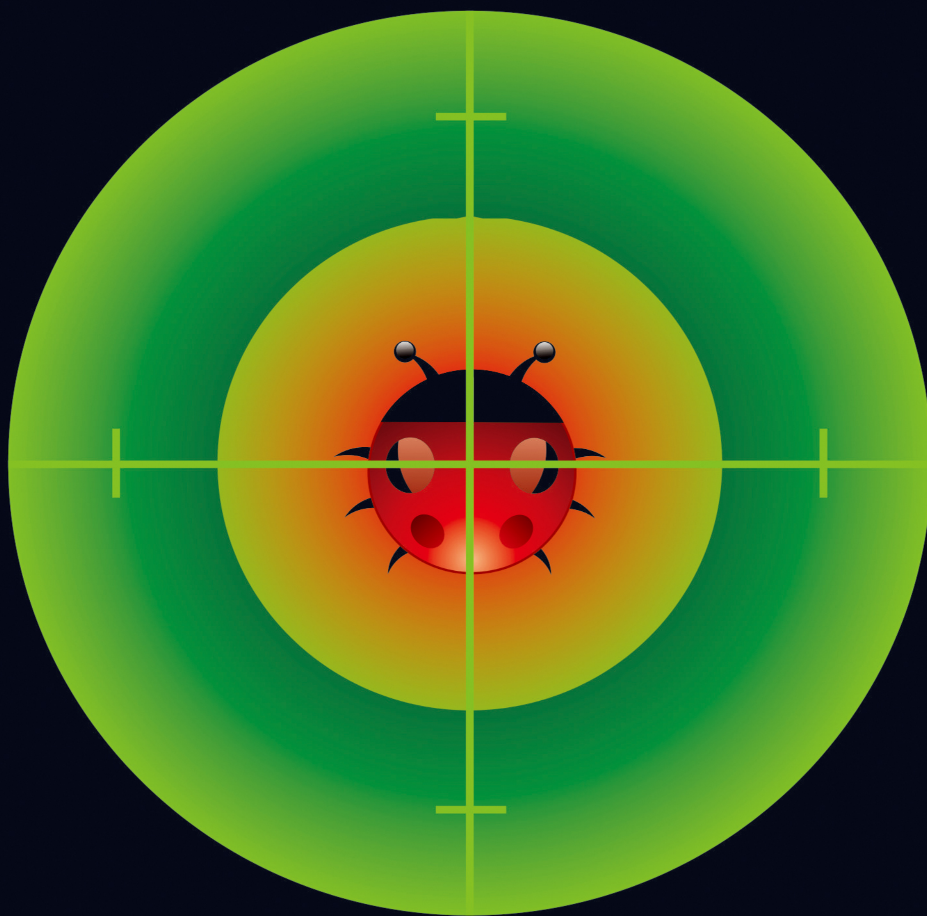
TURING 图灵原创

微软一线测试专家实战精华
全面涵盖软件测试实用技术

软件测试实战

微软技术专家经验总结

史亮 ◎编著





人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明



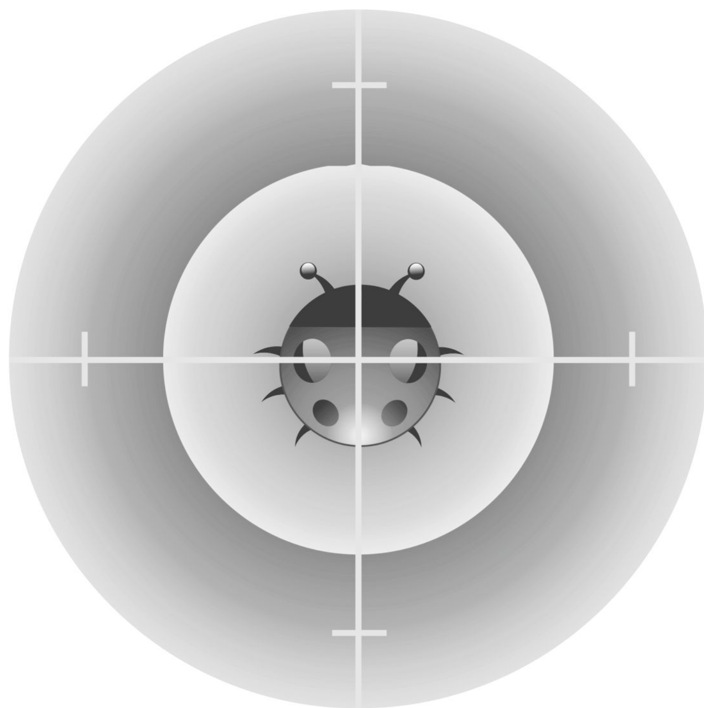
史 亮

毕业于东南大学计算机系，获得计算机软件与理论专业博士学位，研究领域为软件分析与测试。2006年加入微软（中国）有限公司，任职软件开发测试工程师，负责微软在线业务与商业智能产品的测试工作。2011年调至微软总部，从事Microsoft Office 2013的测试工作。2012年与高翔合著了《探索式软件测试实践之路》一书。目前，正从事下一代Microsoft Office产品的研发工作。

软件测试实战

微软技术专家经验总结

史亮 ◎编著



人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

软件测试实战 ：微软技术专家经验总结 / 史亮编著.
-- 北京 ：人民邮电出版社，2014.3
（图灵原创）
ISBN 978-7-115-34584-4

Ⅰ. ①软… Ⅱ. ①史… Ⅲ. ①软件－测试 Ⅳ.
①TP311.5

中国版本图书馆CIP数据核字(2014)第032063号

内 容 提 要

本书从多个角度讨论了测试人员的实际工作，包括缺陷报告、测试文档、测试建模、测试设计、测试自动化、研究产品、研究项目环境、测试管理、个人管理、实践案例等。书中崭新的观念与技术将有助于读者更好地提交缺陷报告，在项目末期的缺陷压力下更好地做回归测试。

本书适用于测试新手以及初级测试人员。

◆ 编　　著 史　亮
责任编辑 王军花
执行编辑 张　霞
责任印制 焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编 100164　　电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京　　　　　　印刷

◆ 开本：800×1000　1/16
印张：22.25
字数：497千字　　　　　2014年3月第1版
印数：1－4 000册　　　　2014年3月北京第1次印刷

定价：69.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第0021号

推荐序一
Recommendation 1

2014年年初，正值学期结束之际，工作比较忙，就利用这次从银川飞往上海的途中，完成了一项非常有意义却很有挑战的任务：为微软总部资深测试工程师史亮博士所著的《软件测试实战》写序。飞机上没有网络、电话，倒是一个很封闭、不受干扰的session，可以集中注意力做好一件事；从另一个方面看，也说明自己对写此序非常重视，希望不愧对朋友所托，能够较好地完成这一“艰巨”的任务。

差不多两年前，我也曾为作者和高翔所著的《探索式测试实践之路》一书写过序，但这本新书有很大不同，涉猎之广几乎覆盖了测试工作各个方面，从缺陷报告、测试建模、测试需求分析与设计到团队建设、个人管理等。从单一测试方式的测试实践走向全面的软件测试实践之路，这期间需要更多的积累，需要更高、更广的视野，由此体现出作者全面的测试知识、独到的思考和丰富的经验。本书不仅内容丰富，入木三分，而且对读者的指导体贴入微，循序渐进地给予具体的示范性操作。例如，通过一周的测试过程指导测试人员如何做好具体的测试工作。本书侧重测试实践，如在讨论“面向调试的测试代码”、“系统测试的测试开发”时，把作者自己实践过的经验一条一条地摆出来展开讨论，并通过列出测试人员容易碰到的问题，逐条解答，以帮助读者解决一些复杂的测试难题，如根据语境（上下文）如何完善组合测试的模型、启发式测试Oracle应用等。本书还介绍了一些新的测试概念或方法，如大规模自动化测试（HiVAT），系统生态图的应用，风险管理中的长回路、短回路等。除此之外，我们还能从本书学习到大量测试想法、清晰的测试逻辑思维、各种测试方法等，极大地增强我们的测试技能。

近几年，作者在美国微软公司总部工作，本书分享的思想、方法和经验多数来自于软件业最发达的欧美，吸收了不少欧美测试大师的卓越思想和优秀实践，引用了近百项资料文献，所有文献都是英文的，时间跨度也很长，从1972年到2012年，长达40多年。可以这样说，本书为我们省去了阅读国外文献的大量时间，仅从这一点来看，这本书也极具价值，更何况本书对一批测试大师的智慧进行了提炼，并应用到实际的案例中。这些测试大师包括我们熟知的James Bach、

Jonathan Bach、Cem Kaner、Michael Bolton、James Whittaker、Elisabeth Hendrickson、Rikard Edgren、Harry Robinson，等等。因此，本书籍此分享了他们贡献给测试知识宝库的一些精华内容，如：

- 产品研究的7大元素 SFDAPOT；
- 启发式测试计划语境（上下文）模型HTPCM；
- 启发式测试策略模型（HTSM）；
- 探索式测试管理方法SBTM（Session-Based Test Management）；
- 测试技术分类体系；
- 敏捷四象限、测试自动化金字塔；
- 基于缺陷模型的快速测试方法；
- 基于部署图的测试指导词；
- 测试Oracle指导词FEW HICCUPPS。

软件测试不同于硬件测试，不仅在于软件的复杂性、软件需求的善变性，而且在于软件的社会性，在一个软件中融入了更多的社会因素，如不同用户的软件应用体验、受业务环境影响的业务操作、受人们心理影响的业务逻辑变化。基于这样的背景，基于上下文驱动测试（Context-driven test）的方法体系引起人们足够的重视，我也深感作者受这个体系（以James Bach、Cem Kaner、Michael Bolton等为代表的流派）的影响比较大，包括对软件测试的理解，更倾向于这样的观点：测试是一种服务测试人员通过服务团队来体现自己的价值；测试是一种技术调查，其目的是向干系人提供有关产品质量的实验消息。本书依旧继承了《探索式测试实践之路》的测试思想，在测试技术这一章，并没有讨论功能测试和性能测试的传统方法和技术，而更多是从“上下文驱动学派”的启发式测试策略、快速测试、探索式测试等方式方法出发，来介绍测试技术，包括情景测试、漫游测试、肥皂剧测试等技术，使测试更有趣，更能适应软件测试的社会性。

本书从缺陷报告开始讨论，虽然编排不同于一般的测试书籍，但也合乎逻辑，缺陷发现和报告是一个测试人的基本能力。基于时间和方法技术的限制，一般应用系统的测试还无法做到百分之百的测试，所以迫使人们常常从反向思维角度思考测试，就是尽可能发现缺陷，缺陷发现越多且能被修复，产品质量就越高。从这一思维方式出发，发现缺陷是软件测试的基本目标之一，或者说，发现缺陷的测试是成功的测试。作者在书中分享了自己的真知灼见，如高质量的缺陷报告来自于高质量的测试。

当敏捷开发席卷世界时，人们对文档重视程度越来越低的时候，本书把测试文档放在很重要的地位，体现了作者的独立思考和务实的态度，不随波逐流，从测试自身要求出发，强调“测试文档是持续演化的工具”，把探索式测试的思想“测试设计、执行、评估（分析）、学习”应用到文档的改进和维护中，并通过从测试计划、设计规约、检查列表到各种形形色色文档的介绍，也让读者更全面、更深刻地理解软件测试。其次，已有测试书籍对测试建模及其应用介绍都比较少，但本书不仅在第4章用一章的篇幅来介绍各种测试建模方法及其应用，而且在第3章、第7章等都有所涉及，揭开了测试建模的神秘面纱，让测试人员不再拒测试建模于千里之外，促进测试人员

在日常测试工作中能够习惯地运用测试建模来解决复杂系统的测试问题，不断地改进测试。

自动化测试永远是软件测试的核心主题之一，本书也不例外，谈到自动化测试策略和实践，如用了较多篇幅讨论面向调试的测试代码、系统级别（包括系统GUI测试）的自动化测试开发。但是，作者用了更大篇幅来研究“产品”和“项目”，因为这些的确是测试的基础，产品是测试的对象，不能真正理解产品、不熟悉产品功能特性和技术实现，测试是不可能做好的；而每个要执行的测试任务都是为了达到测试目标，而测试目标及其实现都直接和其项目的环境因素相关联。

我在阅读本书时，不仅产生了上述这么多的共鸣，而且还有更多的收获和体会。例如，在阅读8.2节内容时，就勾起了我在webex工作的某些回忆。当初我们使用自己开发的项目和缺陷管理系统QAForum，这个系统有一个很大的优点，就是用户可以写SQL语句，构造自己想要的任何形式的、灵活抽取不同数据的缺陷报告。

最后，希望读者在阅读本书时能有更多的思考，并根据本书的指引，阅读相关的资料，进一步地思考、实践、再思考、再实践，一定会受益匪浅。

朱少民

同济大学软件学院教授

推荐序二

Recommendation 2

非常荣幸受邀为史亮的这本新书作序。毫无疑问，史亮是一位优秀的思考者，从本书信手拈来的各种案例中就能够清楚地看到这一点：从离开某项目后其他人接手的困难中，史亮思考和总结出了测试工程师工作的依赖性；从代码覆盖的案例中引申出覆盖率的作用和陷阱……能够在国内的测试领域看到这样一本真正以个人思考为主的原创性图书，实属不易。我相信史亮非常好地贯彻了他在书中"且行且思"这一节中提到的理念：一方面，它暗示软件人员的职业生涯是"漫漫长路"，需要坚持不懈地跋涉；另一方面，它指出持续的努力应该伴随持续的思考，执行与反思缺一不可。

我从1998年偶然进入软件测试领域，在其中投入了十多年的精力，对软件测试的不少方向都有所涉及，期间经历过多种类型的项目，和不同的人员合作过，尝试过各种测试技术和方法。虽然从2011年起软件测试不再是我的主要方向，但我有幸经历了中国的软件测试行业迅速发展的十年，对于软件测试本身有着自己的一些理解和思考。

当然，即使在这个行业投入了十多年的时间，我仍然无法为"该怎么做软件测试"这个问题给出简单明确的回答。从2011年开始，我的主要角色变成了互联网组织的整个研发团队的管理者，这个位置使得我可以不再站在一个纯粹测试者的角度看待问题。站在组织的角度看待测试，最关心的问题不再是"该怎么做软件测试"，而是"软件测试该如何促进组织"。以促进组织（生产率等）为目标的测试，自然不仅仅需要测试工程师的工作，还需要开发工程师、产品经理等各种角色共同参与进来。有代表性的互联网企业（如Google、Facebook）不约而同地采用了小规模的测试团队（甚至不设置单独的测试团队），越来越多的开发工程师开始关注测试和强调测试技能（包括对可测试性的关注）。在这样的趋势之下，现有测试行业的主要从业者——测试工程师自然会面临压力和机遇。压力不言而喻：来自组织和开发者的更高期待使得测试工程师不得不具有更全面的视野、更扎实的技术能力，以及更有价值的产出；而机遇，则是测试工程师可以有机会摆脱一直以来圈住自己的"发现缺陷"的定位，真真正正地为组织创造更大的价值。

然而，“创造真正的价值”需要测试工程师系统的能力，仅靠亦步亦趋地跟随某个流程无法达成；幻想通过盖世神兵（某个工具）或是幻想掉下山崖拿到神秘的内功心法也无法达成。有人说测试工程师是个“越老越吃香”的行业，我却要说，这个行业需要的不是资历，而是“经验”。即使你把手头上的工作重复十年，没有思考，不经过系统的学习和提炼，你的经历永远只能是低层次的重复而已。该如何学习？该如何思考？该如何系统化你的经验？

不得不说，就我看到的状况而言，大部分测试工程师还并未理解这个巨大的转变。仍然有不少人认为测试就是根据流程完成任务，或是仍然执着于测试与开发的界限，画地为牢地守着那些可怜巴巴的“测试技术”。从这个意义上说，史亮这本书适逢其时，它不是一本以说教口吻教你“该如何一步一步做好测试”的书，也不是一本告诉你测试领域又出现了多少新名词的书，它用作者本人的经验，和你分享一个在测试行业勤于思考的人对测试的理解。

这本书不是一本让你用来模仿的书，而是一本让你有所思考的书。史亮在本书中和盘托出的，不是自己的武功招式，而是自己的心法。如果想要从本书中得到最大的价值，我建议你静下心来，仔细品读史亮在本书中娓娓道来的思考，在他的带领下，和他一起领略思考的乐趣吧。

段念

豆瓣工程副总裁

2014年2月16日于北京

推荐序三

Recommendation 3

我和史亮从2009年就认识了，对于史亮在测试技术上的钻研精神我非常敬佩。前两年我有幸和史亮合作一起编写《探索式测试实践之路》一书，在那段测试技术交流的日子里，史亮对于软件测试的前沿测试技术和测试理念的理解让我印象非常深刻。他还积极地在工作中实践最新的测试技术，分享自己的心得体会，一直是我学习的榜样。

通过查看这本书的参考文献就知道，史亮很认真地编写这本关于软件测试技术的实战书籍，我也仔细地阅读了该书相关章节，最大的感受就是这本书的可操作性比较强，与传统测试书籍的理论知识较多而实际案例较少相比，这本书更能让我们体会到理论和实践的相互结合和相辅相成的关系。史亮在微软美国总部接触到了这么多前沿的测试技术和理论并将它们付诸实践，这些宝贵的经验都沉甸甸地包含在这本书里，特别是第4章的建模技术和第5章的测试分析技术，史亮将业界大部分测试技术和分析方法通过简单实用的方式描述出来，方便读者学习和应用。我认为，读者看了这本《软件测试实战》后，80%的软件测试书籍和测试文章都可以不用再去阅读了，因为精华的内容和微软一线测试专家亲自实践的心得都涵盖在本书里。

史亮不仅在测试开发和测试管理上有自己的独特见解，也善于在工作过程中思考如何更好地做软件测试，如何更好地分析产品、提高测试效率。我和史亮都认为，软件测试分析、设计、管理在整个项目周期中是个动态的过程，并且存在较好的手段来提高和控制这个动态过程，《软件测试实战》将告诉读者如何更好地控制和提升整个动态过程。同时这本书也传递了这样一种理念：软件测试不仅仅是测试工程师的个人舞台，更需要和项目团队、被测项目/产品、个人管理充分融合和滋润，才能完美地体现软件测试的价值。

目前国内的很多大公司都有测试开发工程师这个岗位，但是很多人会认为测试开发工程师更多的是从事开发的工作，而不是测试的工作。史亮将自己在微软总部的亲身实践经历告诉读者，

如何来理解测试和开发的关系，如何把测试开发工作效益最大化，如何做有效的自动化测试，而不是盲目的自动化测试。

最后，我极力推荐这本《软件测试实战》，因为它不仅会让你学习到最新的测试技术和实践成果，而且会让你体会和理解一个在测试技术上不断突破自我的工程师的心路历程。

高翔

淘宝资深测试工程师，《探索式测试实践之路》合著者



我在攻读博士学位时开始研究软件测试，毕业后一直任职测试工程师，从事第一线的测试工作，不知不觉已有十余年的光阴。在此期间，我阅读过大量的测试文献，参与过多个不同类型的测试项目，既学到了许多有价值的方法，也观察到了一些不太有效的实践，既通过努力获得过成功的经验，也从错误中得到了宝贵的教训。随着经历的增长，我渐渐构建出自己的知识体系，从实践中打磨出一批指导测试工作的策略与经验。自然而然，一个朴素的想法浮现出来：如果将我的所学所知分享给更多的测试工程师，想必能帮助他们节省学习与积累的时间，以更快地提高测试水平。在该想法的驱动下，我广泛地阅读了测试文献，深入地反思了自己的实践，并开始了漫长的写作。成果就是您所读到的这本书。

本书分享了我从事软件测试工作所学到的知识和所总结的经验，旨在帮助测试人员建立正确的观念，并掌握一批切合实战的测试技术。一方面，我总结了测试专家的见解和方法，将其精华内容综述在本书之中，以帮助读者提高学习效率、快速地掌握综合性的技能。另一方面，我努力将自己的经验和反思融入书稿，使它反映出我在工作中使用的策略、方法和技巧。总之，这是一本注重实效的书，尝试用理论结合实践的方式来解决现实的问题。

本书的组织方式

在概述一些基本测试观点（第1章）之后，本书按照“启发式测试策略模型”（参见4.2.1节）的基本元素来展开论述：观察到的质量（第2章）、测试设计（第3、4、5、6章）、产品元素（第7章）和项目环境（第8章）。随后，讨论了测试人员如何参与团队工作（第9章）并实施个人管理（第10章）。

- 第1章阐述我对软件测试的基本观点，介绍了我的测试价值观。作为指导原则，它们将贯穿全书的内容。

- 第2章讨论了测试人员最主要的工作产出——缺陷报告，介绍了一批实践方法，帮助测试人员高质量地报告缺陷，并利用该过程来改进测试设计。
- 第3章讨论作为测试辅助工具的测试文档，介绍了一些编写和维护文档的原则和方法。通过分析一批具体的测试文档，展示出测试设计的迭代性和多样性。
- 第4章介绍指导测试设计的模型，通过应用组合测试，阐述了如何根据项目语境来完善测试模型，然后介绍了测试建模的基本原则和常用方法。
- 第5章介绍一个测试技术分类系统，以概览各类测试技术。然后，讨论了一批有价值的测试技术，包括启发式测试先知、漫游测试、快速测试、情景测试等。在此基础上，强调了测试人员应该多样地选择测试技术，以动态地优化测试的价值。
- 第6章讨论了测试开发的基本分类，然后针对自动化测试、计算机辅助测试和大规模自动化测试，阐述了它们的基本概念、设计目标、开发策略和实作方法。
- 第7章讨论如何从测试视角来研究软件产品和业务领域，介绍了静态分析、动态分析、关系人研究、需求评审、测试调查、网络调研、领域研究等研究方法。
- 第8章讨论如何从测试视角来研究项目环境，介绍了团队分析、缺陷分析、源码分析、构建分析、自动化测试分析、基于风险的测试等研究方法。
- 第9章探讨测试人员如何有效地在团队中工作，以及如何恰当地实施测试管理。针对一些常见任务，如小组协作、测试计划、工作量估算、软件度量等，提出了一批注重实效的方法。
- 第10章分享了我实施个人管理的基本方法，包含时间管理、个人学习、经验积累、专业发展等。

目标读者

本书的主要读者是具备一定测试经验、想要进一步提高测试能力的测试工程师。全书综合了测试行家的专业建议和我的实践经验，探讨了测试价值观、测试设计、产品研究、项目研究、团队协作、个人管理等多个方面，能够帮助读者更好地理解软件测试，并提高实践水平。本书的内容面向广大的测试社区，并不要求读者掌握特定的背景知识。我希望它能够适合大多数测试人员，并通过解决现实问题来引起读者的共鸣。

此外，本书是很好的软件工程课程、软件测试课程和测试培训的参考资料，能帮助本科生、研究生和测试学员更好地理解真实的软件测试。一些测试人员在工作后会发现，课程教材和培训教程所描述的内容并不切合实际的测试工作。例如，课本中的测试设计和执行是线性实施的，真实测试流程却是迭代展开的；缺陷报告常常是测试工作的核心产出，教材却没有讨论如何有效地管理缺陷报告；测试人员被要求撰写文档，其成果又被束之高阁，却少有教程讨论如何编写有价值的文档。这时，他们可以参考本书所提出的建议，从而发展出更有效的工作策略。

如何阅读本书

本书第1章讨论软件测试的基本事实和价值观，是全书内容的基础，需要首先阅读。在读完第1章后，读者可以按任意顺序阅读本书。您既可以顺序浏览，以概观软件测试，也可以有选择地阅读感兴趣的章节，并在阅读的过程中参考相关内容。

这是一本关于实践的书，许多建议和方法来自于实践。软件开发专家Ralph E. Johnson 指出，“从实践中来的知识在没有实践之前是无法被真正理解的”（practical knowledge has to be experienced to fully understood），测试专家Cem Kaner、James Bach等也认为“你不能掌握测试，除非你重新发明它”（You can’t master testing unless you reinvent it）。在阅读过程中，读者需要积极思考本书的方法是否适用于自己的项目，然后将恰当的方法应用于真实的测试，并认真评估其效果。通过练习、评估和反思，读者能够掌握方法的原理和细节，并混入自身经验和其他技术，以演化出新的方法。坚持这样的研究和创新将帮助测试人员走上精通之路。

致谢

作为作者，我诚挚感谢人民邮电出版社和北京图灵文化发展有限公司为我提供创作的平台。感谢本书的编辑王军花、张霞等，你们的专业态度和细致工作提升了本书的品质。

我还要感谢审稿人和推荐人的批评指教，你们的细心评阅和宝贵意见令我受益匪浅。谢谢你们的时间和智慧。

最后，我要感谢妻子汤红红，没有你的理解和支持，我无法完成此书。我还要感谢儿子史泰德用欢笑带给我快乐与能量，你令我想成为一个更好的人。感谢你们让我梦想。

目录 CONTENTS

第1章

软件测试基础

本章将阐述一些软件测试的基本观点，它们是全书的基石。虽然它们并没有得到所有人的认可，但是我认为它们反映了软件测试的基本事实。测试人员需要仔细分析它们，并用于实践，才能有效地实施软件测试。

1.1 软件的复杂度已经超越了人的理解能力

必应词典[①]是我常用的一款桌面软件（如图1-1所示），它将用户输入的英文单词发送给必应服务器，然后接受服务器所返回的中文解释，最后将中译显示在界面上。



图1-1 必应词典

① http://dict.bing.msn.cn。

必应词典是一个.NET程序[①]。它会启动一个专属线程来接受服务器的返回结果，其托管代码（managed code）的调用栈类如代码清单1-1所示。

代码清单1-1　必应词典的调用栈

```
System.Net.UnsafeNclNativeMethods+OSSOCK.recv(IntPtr, Byte*, Int32,
    System.Net.Sockets.SocketFlags)
System.Net.Sockets.Socket.Receive(Byte[], Int32, Int32,
    System.Net.Sockets.SocketFlags, System.Net.Sockets.SocketError ByRef)
System.Net.Sockets.Socket.Receive(Byte[], Int32, Int32,
    System.Net.Sockets.SocketFlags)
System.Net.Sockets.NetworkStream.Read(Byte[], Int32, Int32)
System.Net.PooledStream.Read(Byte[], Int32, Int32)
System.Net.Connection.SyncRead(System.Net.HttpWebRequest, Boolean, Boolean)
System.Net.Connection.PollAndRead(System.Net.HttpWebRequest, Boolean)
System.Net.ConnectStream.PollAndRead(Boolean)
System.Net.HttpWebRequest.EndWriteHeaders(Boolean)
System.Net.HttpWebRequest.WriteHeadersCallback(System.Net.WebExceptionStatus,
    System.Net.ConnectStream, Boolean)
System.Net.ConnectStream.WriteHeaders(Boolean)
System.Net.HttpWebRequest.EndSubmitRequest()
System.Net.HttpWebRequest.CheckDeferredCallDone(System.Net.ConnectStream)
System.Net.HttpWebRequest.GetResponse()
System.Web.Services.Protocols.WebClientProtocol.GetWebResponse
    (System.Net.WebRequest)
System.Web.Services.Protocols.HttpWebClientProtocol.GetWebResponse
    (System.Net.WebRequest)
System.Web.Services.Protocols.SoapHttpClientProtocol.Invoke(System.String,
    System.Object[])
Engkoo.Ehc.Core.EhcAPI.EhcDataServiceV1.GetEditModeResult(System.String,
    System.String, System.String, System.String, Int32, Boolean)
Engkoo.Ehc.Core.OnlineDictionaryV2.ProcessEditRequest(DataRequest)
Engkoo.Ehc.Core.OnlineDictionaryV2.AsyncProc()
System.Threading.ThreadHelper.ThreadStart_Context(System.Object)
System.Threading.ExecutionContext.runTryCode(System.Object)
[HelperMethodFrame_PROTECTOBJ: 0da8f3f8]
    System.Runtime.CompilerServices.RuntimeHelpers.ExecuteCodeWithGuaranteedCleanup
    (TryCode, CleanupCode, System.Object)
System.Threading.ExecutionContext.RunInternal(System.Threading.ExecutionContext,
    System.Threading.ContextCallback, System.Object)
System.Threading.ExecutionContext.Run(System.Threading.ExecutionContext,
    System.Threading.ContextCallback, System.Object)
System.Threading.ThreadHelper.ThreadStart()
```

该调用栈涉及26个函数，使用了一些典型的开发技术，具体如下所示。

- Socket通信（`System.Net.Socket`）
- HTTP通信（`System.Net.HttpWebRequest`）
- SOAP通信协议（`System.Web.Services.Protocols.SoapHttpClientProtocol`）

① 本章讨论的是必应词典1.7版。自2.0版开始，必应词典的主程序（BingDict.exe）不再是.NET程序，而是C/C++程序。

- 异步调用（`Engkoo.Ehc.Core.OnlineDictionaryV2.AsyncProc`）
- 线程（`System.Threading`）

随着开发技术的发展，现在开发人员不必理解这些技术细节，只要调用程序库，就可以完成客户端（必应词典）与服务端（必应服务器）的通信。相比于底层技术，这些程序库提供了更高的抽象层次。然而，软件专家Jole Spolsky提出了经验法则"抽象漏洞定律"："所有非平凡的抽象，在某种程度上，都存在漏洞。"[Spolsky02]他敏锐地指出，虽然高抽象层次的语言、程序库和框架等技术提高了开发者的生产力，使他们不必总是关心技术细节，然而总是存在一些情况，需要开发者深入底层，去研究那些为了提高生产率而省略的细节问题。高抽象级别的技术虽然可以减少编写代码的时间，但是并不能减少开发者学习整个技术栈的时间。只停留在高抽象层次的开发者将难以解决复杂的实际问题，且有可能引入更多的设计错误。

对于必应词典，与服务器通信是相对简单的任务，它使用了一些常见的重要技术，开发者完全理解这些技术需要付出很大的努力。然而，必应词典所涉及的技术细节远不止于此，它拥有40多个线程，加载170多个动态链接库，拥有1000多个Windows句柄。毫不夸张地说，几乎没有人可以完全理解它使用的所有技术，也无法掌握它任意时刻的状态。

不过，软件技术只是软件复杂性的一部分。成功的软件必须帮助用户解决实际问题，使他们获得成功。因此，项目团队需要研究用户情景和领域知识，然后创造性地提出解决方案，并选择合适的技术来实现。在这个研究、创造、应用的过程中，项目团队会面临更多的挑战和困难，而能否处理领域复杂度和设计复杂度是决定产品成败的关键。例如，对于必应词典而言，发送请求并接受响应是相对容易的，困难的是为一个单词提供准确的定义、例句、搭配、同义词等信息，更困难的是为一个句子提供恰当的翻译，并流畅地将它朗读出来。这些困难的任务恰恰是用户选择词典软件的重要参考。

由以上讨论不难看出，软件复杂度包含技术、领域、设计等多个方面。随着软件行业的快速发展，软件复杂度已经超越了个人的理解能力。那么，从测试的角度来说，测试人员应该如何应对这一情况呢？本书所介绍的大部分实践都是为了解决该问题的。在这里，我简单陈述一些基本态度和方法。

- 对于复杂的软件，任何人都不可能掌握全部的信息。如果测试人员对软件的理解存在许多偏差，他的测试策略一定会包含错误。为了更好地理解软件，他需要与产品经理、程序员、领域专家、测试同事等协作，还需要研究项目文档、技术资料、领域专著等文献，并通过实际测试去获得第一手知识。测试人员不应该依赖单一的或局限的信息源，他应该从各种渠道获得信息，多角度地研究软件。
- 由于软件如此复杂，大多数测试人员在项目之初都不甚了解软件，其拟定的初始测试方案或多或少都存在错漏，有些甚至存在严重错误。注重实效的测试人员，会承认测试方案是不完备的，并迭代地实施测试，通过持续地评估和反思来逐步增强测试方案。

- 对于现实世界的软件，穷举测试是不可行的。任何实用的测试技术都是基于一定策略的采样，即从无限多的测试输入中选择一个数量有限的子集，通过运行这批测试来评估软件的整体情况。考虑到软件的复杂度，任何一种测试技术都存在不能正确评估软件的风险。而且，在测试之初，测试人员也很难预料究竟哪些测试技术才适合当前项目。为此，测试人员应该积累多种测试技术，综合运用它们，并随着项目发展积极调整测试策略，才能避免重大测试遗漏。
- 人们处理复杂问题的常用策略是“分而治之”。在软件领域，建立合理的抽象模型是应对软件复杂性的常见方式。在测试领域，测试人员可以建立产品的模型来帮助测试。在建模过程中，他们以当前测试目标为指导，省略无关的产品细节，突出重要的产品元素。面对简化后的产品模型，可以更有针对性地实施测试设计。
- 在大规模软件中，对于少量代码的变更都不可以掉以轻心，因为修改者、代码评审者、测试者都很难预料这段代码被调用的语境和执行后的影响。例如，程序员要修改网页浏览器（如IE、Chrome、Firefox等）排版引擎的一个排版缺陷，通常需要运行大规模的回归测试。因为全世界的网页千差万别，很难预测这段代码会排版何种网页，也很难预测这段代码是否会破坏某些精心排版的页面。只有运行大规模测试，才能保证代码变更的正确性。
- 单凭人的脑力已经难以应对软件的复杂度了，测试人员需要考虑利用自动化测试开发强力的测试策略。

1.2　软件测试是获取信息的技术调查

不同的人有不同的背景、不同的目标、不同的任务，对软件测试也会给出不同的定义，因此不存在放之四海皆准的软件测试定义。不过，对于软件测试工程师而言，我认为有3个软件测试的定义很有启发性。

定义1：测试是为了发现错误而执行程序的过程[Myers79]。

该定义是Glenford J. Myers在1979年提出的，历经30多年的考验，至今仍被广泛引用和讨论。这是一句简短而有力的论述，它紧扣测试的基本活动——执行程序和寻找错误，符合大多数测试人员的工作内容，因此更容易获得广泛认可。此外，它揭示了软件测试的根本原因，即某些重要的东西可能出错[Kaner01]，软件测试就是要发现这些错误。

它给测试人员的启示是，测试人员应该始终质疑并挑战软件。有些测试任务的字面内容类似于“检查本次代码变更是正确的”，好像要求测试人员去验证软件行为是正确的。对于此类任务，测试人员应该假定代码变更引入了未知的错误，然后想尽办法去攻击软件。即便测试人员最后的结论是“我没有发现错误”，攻击性的测试过程也使这个结论更有可信度。

定义2：测试是一个获取信息的过程，用来降低决策风险[Weinberg08]。

Genald M. Weinberg认为人不是完美的思考者，面对复杂的情况会作出许多错误的决定。在

采购、选择、使用、开发软件的过程中，人们需要作出许多决定。不完美的思考导致有风险的决策，可能造成巨大的损失。他指出测试可以获得更多的信息，使人们更周全地思考，从而降低错误决策的风险。

该定义给了我（一个测试人员）一系列启示。

- **测试是服务性的工作**。通过测试，我向整个团队提供关于产品质量和项目环境的信息，帮助他们作出决定。
- **低质量的信息不但无助于决策，还可能浪费团队的时间，甚至作出错误的决策**。因此，我应该总是提供高质量的信息。这通常体现为及时地交流、周密地测试、仔细地报告。
- **团队的决定可能与我的期望不一致，要去理解导致该决定的其他信息**。例如，我提交了一个严重的缺陷，团队领导却决定不予修复。此时，应该提供更多的信息，据理力争，同时要去理解不予修复背后的原因，从而更好地理解软件和项目。
- **为了更全面地提供信息，除了运行软件，我还需要更多的获取信息的方法**。常见的手段包括与程序员交谈、研究项目资料、学习领域知识、阅读源代码、在调试器中观察软件等。
- **除了软件缺陷，测试还可以提供关于项目环境的信息**。例如，在项目过程中，产品安装总是遇到许多问题。在报告缺陷的同时，我还可以向团队领导反馈："开发小组是否在产品安装上投入了足够多的资源？因为安装是任何测试的先决条件，安装的错误将阻碍或延缓测试，测试小组建议将它视作第一等的功能需求。"

定义3：软件测试是一种技术调查，其目的是向关系人提供有关产品（软件、系统或服务）质量的实验信息 [Kaner06]。

该定义是Cem Kaner教授提出的，他也认为测试是一种服务。服务的客户是项目关系人，服务的内容是提供产品质量的实验信息。质量就是对某个（某些）人而言的价值 [Gause89]。不同的人对于质量有不同的评判标准，对于信息有不同的需求。常见的项目关系人包括客户（购买产品的人）、用户（使用产品的人）、程序员、产品经理、运维人员、市场营销人员、管理人员等，他们对信息的需求是测试人员主要的工作内容。请看如下案例。

- 测试经理对测试人员说："你刚提交了一个严重的缺陷，做得好！但是，现在临近发布，修复它的风险很高。你能不能调查一下这个缺陷在哪些情况下出现，会影响哪些用户，导致的最坏结果是什么？我们会根据这些信息来决定是否修复和修复策略。"
- 程序员对测试人员说："我完成了一个补丁的开发，下周将部署到产品环境中。你能否在预发布环境中部署它，并跑一些系统测试？它只是缺陷修复，没有引入新功能，应该不会破坏当前系统的运行。"
- 产品经理对测试人员说："我们在当前版本中启用了新的图形渲染引擎，能够明显提高画质，但是试用者反映这个版本的性能不如上一版。你能不能做一些实验，看看当前版本在哪些场景中比较慢，慢多少？"

- 运维人员对测试人员说："一个用户在产品论坛上发帖说，软件在启动时会弹出一个报错对话框，跟帖的几个人也说遇到过相同的情况。我从没有见过该问题，所以不知如何解决。你能不能看一下这个帖子，试着在测试环境中复现？这个缺陷在下一版已经修复了吗？"

以上案例都表明，除了发现缺陷外，测试人员还通过多种方式向不同的关系人提供信息：向管理人员提供调查报告，向程序员提供代码的质量反馈，向产品经理提供技术支持，向运维人员提供已知缺陷的信息，在邮件组中回答用户提问等。为了更有效地测试，测试人员需要与测试经理讨论，设定测试服务的优先级，为关系人提供满足其要求的高质量的信息。例如，提供给测试经理的信息要面向项目风险，提供给程序员的信息要包含更多的技术细节，提供给产品经理的信息要侧重于用户体验和产品价值，提供给运维人员的信息要建议可能的解决方案。

此外，Cem Kaner还强调软件测试的主要工作是技术调查，即以职业的态度、专业的技能对产品的未知领域进行探索。与地质勘查、罪案侦查、事故调查等调查活动相似，测试具有系统性、客观性、探索性和机动性。

- **测试应该系统地调查被测试对象**。测试人员需要对产品进行周密的分析与检查。地质勘查结束后，勘查人员会给出详细的地质图，以描绘该地区的全貌。测试结束后，测试人员也应该全面掌握被测试对象的情况。
- **测试所提供的信息应该来自科学实验和中立观察**。在真实和客观的基础上，包含合理的推测和建议。
- **刑侦人员会利用多种方法，从各个信息源收集情报，因为很难预测何处会有重大发现**。当突破性情报浮现时，他们会转移工作重点，顺着新线索紧追不舍。测试人员也需要综合运用多种技术和工具，去探究新的信息，并根据新发现及时调整测试方向。
- **优秀的潜水员既能够通过浮潜去游历宽阔的水域，也可以利用水肺潜水去探索深海水域**[DeMarco08]。测试人需要交替使用广度调查和深度调查，从而更有效地提供信息。广度调查有助于发现高风险的区域，深度调查能够提供详细的信息，以支持项目关系人作出正确的决定。

1.3　测试是迭代过程

测试是一个贯穿项目周期的调查过程。为了阐述技术细节，许多测试文献只讨论了该过程的一小部分。因此，它们无意中将测试描绘为一个线性过程：测试人员获得测试任务，对被测软件进行分析，制订测试计划，确定测试用例，然后……没有"然后"了，因为大多数文献只写完第一轮测试设计便结束了。

但是测试并不是一个线性过程。《计算机软件测试》[Kaner01]第1章的目录更符合测试人员的工作方式。

1. 第一个测试周期
   1.1　第1步：从显而易见的简单测试开始
   1.2　第2步：记录还需要测试什么
   1.3　第3步：检查有效用例并观察发生了什么
   1.4　第4步：做一些“快速”测试
   1.5　第5步：总结对程序和问题的认识
2. 第二个测试周期
   2.1　第1步：在进行任何测试之前应该仔细评审对问题报告的反馈，以确定哪些部分值得深入测试
   2.2　第2步：评审对不予修复的问题的意见，它们可能建议进一步的测试
   2.3　第3步：找出上次的测试笔记，加入新笔记，并开始测试
3. 后续测试周期中可能会发生的事情

该目录列举了两个测试周期中测试人员的典型活动。测试周期是一个相对于测试人员而言的概念，它包含获得可测试的新版本、测试、报告问题（即bug）、反思总结等活动。无论项目采用何种开发模型，测试人员总是一个版本接一个版本地测试，其测试活动总是迭代向前的：测试版本1→提交缺陷→修复缺陷→测试版本2→提交新缺陷→修复新缺陷→测试版本3→……

即便在测试周期内部，测试活动也是迭代的。以“第一个测试周期”为例，测试人员不熟悉软件，所以第1步是运行一些简单的测试，来了解软件的行为。在第2步和第3步，测试人员根据第1步获得的信息设计了更多的测试，并记录在案。他使用了一些经典的测试设计方法，如边界值分析和等价类划分。在第4步，他使用了一些启发式测试方法对软件进行快速攻击。在第5步，他对被测软件和测试策略进行了反思，以挖掘软件的风险和测试的不足。可见，整个测试过程是螺旋向上的：初始的测试提供了正式测试设计所需的信息；正式的测试用例较系统地检查了系统；当正式测试揭示出软件风险时，快速测试机动地探索相关领域，以确定是否存在严重的问题，并为后续的正式测试提供信息。

后续的测试周期会利用先前测试所产生的信息。以“第二个测试周期”为例，测试人员分析对问题报告的反馈，了解团队中其他人对软件行为的看法。利用这些新信息，他可以确定测试的重点。他还会参考第一个周期的测试笔记，但是不会照本宣科地重新执行一遍，而是利用新信息设计新的测试。从这个角度来看，测试类似于移动的军舰向另一艘移动的军舰开火。开火的军舰在移动是因为测试人员对被测软件的认识在不断地增强，他知道哪些测试不太容易找到错误，哪些测试更可能找到问题。目标军舰在移动是因为程序员在持续提交新功能和修复缺陷，导致软件也在不停地变化。因此，测试策略也需要保持动态变化，使得炮口一直瞄准飞驰的目标。

迭代的最大优点是能够快速获得测试设计的反馈，从而逐步完善测试设计。程序员已经认识到软件开发过程会引入大量的错误，应该利用快速反馈来发现设计中的错误。因此，测试驱动开发、结对编程、持续集成、自动化测试等技术得到了普遍认可和应用。测试设计与软件设计相似，

都是高智力的创造性工作，也可能会产生错误的设计。测试人员也需要快速获得测试设计的反馈，有意识地利用迭代可以为此奠定良好的基础。

1.4　测试人员的工作效率取决于他对软件和项目的理解，而不是他掌握的测试技术

我曾经长期测试一个网络应用。当我离开这个项目时，测试经理安排一个测试员工来接替我。他刚刚入职，对被测软件和业务领域都不了解，在工作中遇到了许多困难。在我加入新团队后，我还通过电话数次回答了他的问题。后来，我反思了当时的情况。作为一个测试工程师，我的工作效率明显优于接替我的同事，主要原因包括以下几点。

- **我理解产品**。我知道它的业务目标，了解它通过什么方法去实现目标。因此，我能够快速地制定测试方案。
- **我理解用户的期望**。我知道哪些功能绝对不能出错，需要仔细测试，也知道哪些功能允许一些瑕疵，即便出错，也可以在下一个版本（通常在3个月之后发布）中修复。因此，我能够更好地分配测试时间。
- **我理解产品的架构**。通过阅读产品源代码，我知道哪些模块容易出现哪些缺陷。因此，我可以针对不同的模块采用有针对性的测试策略。
- **我知晓如何回避浪费时间却没有收益的任务**。例如，我曾经尝试用自动化测试用例去测试用户界面，但是发现此类自动化测试很不稳定，需要很高的维护代价，却不能发现错误。于是，我只为Web服务编写自动化测试用例，用手工测试来测试用户界面。
- **我了解产品元素和项目团队**。当出现缺陷时，我知道如何阅读系统日志发掘蛛丝马迹；当我遇到困难时，我知道向哪位程序员或测试人员求助。因此，我可以深入挖掘并快速推进。
- **我在原先的团队工作了很长的时间，与同事建立了良好的关系**。当我提出一些可测试性的建议时，比较容易得到程序员的支持。

从以上几点不难看出，我能够更有效地测试，其主要原因不是我掌握更多的测试技术，而是我更了解软件产品、业务领域和项目环境。通过逐点分析，可以得到如下启示。

- 产品是一种解决方案，如果没有解决问题，它就是无用的[Kaner12]。测试人员需要了解软件产品和业务领域，才能设计有效的测试。
- 测试是一种信息服务，要了解服务对象的需求。如果用户不能容忍某些错误，测试人员就需要仔细测试相关功能；如果用户对一些瑕疵并不在意，测试人员就不必在此花费过多的时间。只有了解服务对象的优先级，才能更好地设定测试工作的优先级。
- 不同的模块采用不同的技术，拥有不同的典型错误。只有了解软件实现，才能设计差异化且有针对性的测试用例。
- 测试设计可能包含错误，测试人员需要从错误中吸取经验和教训，避免重蹈覆辙。

- 当测试工作遇到困难时，测试人员需要知道从哪里寻找信息。了解被测产品和测试工具能够提供的信息，了解哪位同事知道更多内幕，会节省时间。
- “人脉”有时候会极大地提高测试人员的工作效率。测试人员需要与程序员和测试同事保持良好的关系。达成协作关系的关键之一是测试人员能够为同事们提供高质量的信息服务。
- 在职业生涯中，测试人员总是会遇到新的软件、项目和团队。他应该养成一种好的思维方法和测试风格，以便快速地学习并理解产品和项目。

实施高效的测试需要很多条件。熟练地掌握测试技术是一个很重要的因素，但很少会是决定性的因素。只有充分掌握软件产品和项目环境，测试技术才能找到大放光彩的舞台。

1.5 小结

本章介绍了软件测试的基本价值观。作为指导原则，它们将贯穿全书。

- 软件复杂性来自于领域、设计、技术、开发过程等多个方面。为了应对高度复杂且持续变动的软件，测试人员需要从多个来源收集信息，并在项目全程收集对测试设计的反馈。
- 软件测试的基本目标是发现软件错误，并予以修正。
- 软件测试是一种信息服务。以技术调查和科学实验的方式实施软件测试，能够提供高质量的服务。
- 无论测试人员是否意识到，软件测试都是迭代展开的。有意识地利用软件测试的迭代性，可以更好地测试。
- 高效的软件测试要求测试人员理解软件和项目的方方面面。学习与实践要伴随测试全程。

第 2 章

缺陷报告

在许多项目环境中，缺陷报告是测试人员最主要的工作产出，是将测试人员和项目团队广泛联系在一起的纽带。

- 程序员会阅读缺陷报告，以了解缺陷的症状和重现步骤。好的缺陷报告能帮助他快速地定位问题；差的缺陷报告会浪费他的调试时间。
- 产品经理会阅读缺陷报告，以了解缺陷的症状和严重性。好的缺陷报告准确地传递了用户质量的信息，帮助他设定修复优先级；差的缺陷报告会误导他作出错误决定，甚至将一些严重的缺陷标记为“不予修复”。
- 在一些团队，产品经理、开发经理和测试经理会举行缺陷评审会议，对缺陷是否修复进行“最终判决”。好的缺陷报告会提高会议效率；差的缺陷报告会降低会议效率，甚至让评审小组作出错误的决策。
- 在一些大型项目中，缺陷报告是测试小组以外的人了解测试人员工作的主要途径。他们会根据缺陷报告的质量评价测试人员的工作能力和职业素养。

可见，测试人员应该认真对待缺陷报告，因为它们关乎整个团队的工作效率和测试人员的个人声誉。编写高质量的缺陷报告是测试人员最重要的基本功之一。本章将介绍一些编写和处理缺陷报告的方法，帮助测试人员更有效地与项目团队交流。

2.1 报告缺陷是为了让缺陷得到修复

报告缺陷的首要目标是“使正确的缺陷得到修复”[Kaner08a]。所谓“正确的缺陷”，是那些值得修复的错误、局限、失败等降低软件价值的问题。测试人员面临的挑战“对软件价值的影响”和“是否值得修复”都是主观判断，不同的人基于不同的背景和立场很可能得出不同的结论。测试人员需要说服程序员和缺陷评审小组，使他们相信修复当前的缺陷是正确的。因为测试人员通常不负责修复缺陷，为了使缺陷得到修复，需要提交和维护高质量的缺陷报告，将正确的信息传递给团队。这需要测试人员持续地努力和长期地积累。在此过程中，以下策略会有所帮助。

- 清楚地说明此问题对用户价值的危害。

- 提供尽可能多的技术信息，方便程序员调试。
- 尽早提交缺陷报告。
- 报告发现的所有缺陷，即便有些缺陷难以重现。

第一，缺陷报告要重点说明该缺陷对用户价值的损害。许多软件项目都进度紧张、资源有限，项目团队修复一个严重的缺陷，可能需要考虑多个因素。例如：修复该缺陷可能需要2天，而实现一个用户建议的新功能也需要2天，是修复缺陷还是增加功能对用户更有价值？目前，软件即将发布，修复该缺陷会不会引入新的问题？为了修复该缺陷而推迟发布是否值得？这些问题的核心因素是该缺陷是否显著地降低了软件对用户的价值，以至于开发人员与测试人员愿意占用开发活动的时间、冒着引入更多缺陷的风险来修复它。如果缺陷报告没有写清楚缺陷的影响，产品经理或缺陷评审小组有可能会错误地认为这是一个不重要的问题，并将它标记为“不予修复”。

为了让缺陷评审者了解真实的情况，测试人员需要从多个角度思考如何有效地传递缺陷信息。

- **测试人员在提交缺陷时最好使用自制的缺陷模板**。缺陷模板为一些常用缺陷字段提供了默认值，减少了输入的工作量。更重要的是，缺陷模板可以提供缺陷报告的结构，提醒测试人员输入必要的内容。例如，缺陷模板可以包含“用户影响”一节，提醒测试人员输入缺陷对于用户价值的危害。如果缺陷对用户的影响非常明显，测试人员可以忽略该节，但这是测试人员思考后的决定，而不是由于忙碌忘记填写。
- **要为使用软件的人“仗义执言”**。测试人员很可能是第一个完整使用软件的人，能够发现一些用户在日常使用中遇到的问题。某些问题不是严格意义上的计算错误（例如某个常用按钮没有键盘快捷键），某些问题来自于开发技术的限制（例如32位程序只能利用2GB的内存，一些需要大量内存的操作可能因为该限制而失败），某些问题在以前的版本中也存在。无论如何，这些问题使用户受到挫折，降低了软件对他们的价值。测试人员需要报告这些问题，并阐述它们给用户带来的困扰。
- **要“放大格局”，软件的“关系人”还包括程序员、测试人员、运维人员、销售人员等。**如果软件存在一些问题或局限，使得他们的生活变得困难，测试人员也应该提交缺陷报告。例如，如果测试人员发现被测的网络服务没有记录足够的日志，以致整个业务流程类似于一个黑盒，测试检查变得很困难。他应该报告该问题，建议服务日志还需要记录哪些内容，并解释理由。测试人员可以指出，从长远看更多的日志不但能够帮助测试人员更快地发现错误，而且有利于运维人员发现在线系统的问题，也有利于程序员根据日志定位问题，从而提高整个团队的效率。
- **为了提供有说服力的证据，测试人员可能需要做一些“研究”**。例如，测试人员发现一个杀毒软件兼容性问题——在被测软件执行特定操作时，某个杀毒软件会报告发现病毒。为了评估该问题的严重性，他需要分析该操作的重要性和使用频率，可能的手段包括阅读规格说明、查询用户日志、询问产品经理等。此外，他还可以通过搜索引擎查询该杀毒软件的市场份额，估算有多少用户可能遇到该问题。

- **测试人员还可以寻找“专家”支持**。所谓专家是受到整个项目团队认可和信任的人，他的意见会有更大的影响力。他可能是一位资深员工，经历了多次产品发布；也可能是一位领域专家，对业务领域有深刻的理解；还可能是一位用户代表，能够提供真实的用户反馈。测试人员可以与他讨论所发现的问题，获得他的支持。如果他可以在缺陷报告中写下有份量的评论，或在缺陷评审会议上支持修复问题，该缺陷将很可能得到修复。

第二，缺陷报告应该提供尽可能多的信息，以便程序员修复缺陷。缺陷不被修复或没有被及时修复的常见原因是程序员没能重现该缺陷。对于难以重现的问题，测试人员应该收集并报告尽可能多的信息，例如有可能重现缺陷的操作步骤、屏幕截图、视频录像、所使用的数据文件、软件的内存转储、追踪日志、网络通信记录等。软件测试是技术调查，缺陷报告作为调查结果，应该包含业务层面的信息，让缺陷评审者了解软件价值的损失，还应该包含技术层面的信息，帮助程序员更快地修复问题。

第三，缺陷越早被报告，越有可能被修复。当项目临近发布时，项目团队会对代码变更采取谨慎的态度，因为最后时刻的代码修改可能会引入一些新的缺陷，而短时间的测试不能发现这些缺陷。因此，缺陷评审会议常常因为风险较高而拒绝修复一些缺陷。这样的情况我遇到过多次，如果能早一些提交该缺陷，哪怕是早一个星期，它就能得到修复。这给我两个启示：一是要第一时间提交发现的缺陷；二是在规划测试时，要使测试策略能够尽早地发现严重的缺陷。

第四，为了提供完整的信息，测试人员应该报告发现的所有问题。这并不意味着提交重复的信息。在提交缺陷报告之前，测试人员可以查询一下缺陷管理系统，了解被测功能有哪些活跃的缺陷。如果其他测试人员已经提交了这个缺陷，那么测试人员可以跳过报告，继续测试。我建议测试人员在查询相似缺陷上不必花费太多的时间，只要避免明显的重复提交即可，因为重复提交的开销远小于遗漏报告的代价。而且，程序员通常能够快速地链接根源相同的两份缺陷报告，将它们“并案”处理。

有时，软件的表现出乎测试人员的预料，但是他并不能确定这一定是个缺陷。这说明测试人员对软件的设计和实现还有不了解的地方，他应该将此疑惑视为一个学习的机会，通过阅读文档、咨询同事等方法来获得解答。然而，在测试时间紧张的情况下，深入调查是不适宜的。如果不能立即获得解答，测试人员应该提交缺陷报告，让产品经理或缺陷评审会议来回答。

在提交缺陷报告时，测试人员面临的最大挑战是那些难以重现的缺陷。大多数测试人员都有这样的经历：测试执行时，无意中发现一个缺陷，于是再测试一遍，来确认重现步骤，这时该缺陷却“神奇地”消失了。这说明某些未知的因素在起作用，它们改变了软件的行为，使得缺陷不能重现。这时，测试人员的任务是用各种手段去发现这些因素，让缺陷重现。不幸的是，重现缺陷的努力常常会失败。面对这种情况，测试人员仍旧应该提交缺陷报告。他应该在报告中坦承该缺陷不能稳定重现，然后报告他知道的所有信息，例如为了重现缺陷做了哪些实验、使用了哪些实验数据、实验结果如何、对于缺陷根源的猜测等。

2.2　高质量的缺陷报告来自于高质量的测试

测试是一个迭代的过程，缺陷报告是其中的一环。它既是一段测试的结束，也是一段测试的开始。测试人员应该让缺陷报告和测试执行相互支持，让缺陷报告来推动更深入的测试，让高质量的测试产生高质量的缺陷报告。

2.2.1　分配测试时间

在提交缺陷报告时，测试人员常常需要考虑如何使用接下来的测试时间。

- 是继续调查当前缺陷，以提供更简化的重现步骤，还是开始新的测试？该问题需要测试人员平衡调查时间和测试时间。用更多时间去调查可能提供更清晰的缺陷报告，从而简化缺陷评审者和修复者的工作；用更多时间去测试可能更早地发现其他缺陷。
- 是继续测试该功能（或情景），还是测试其他功能（或情景）？该问题需要测试人员平衡所提供信息的深度和广度。继续测试可能提供更深层次的信息，转移测试可能提供更广泛的信息。

这两个问题的本质是测试时间是重要但有限的资源，测试人员需要合理地分配该资源，以提供“足够好”的信息。“足够好”是指“有足够的信息可供客户作出好的决策”[Kaner01]。测试服务拥有许多客户，他们对“足够的信息”有不同的期望。例如，程序员希望缺陷报告尽可能详细，测试经理则希望能够更早地测试软件的方方面面。测试人员需要综合考虑各方面因素，作出合理的判断。

由于软件项目千差万别，并不存在普遍适用的分配测试时间的指导方案，但存在一些启发式方法来简化问题。测试人员可以考虑使用“时间盒”来管理自己的时间。

例如，测试人员用一个大型的数据文件发现了一个软件错误。他认为删去文件中的一些无关数据，也能重现该错误。精简的数据文件会使缺陷报告更直接明了，但是需要投入时间。这时，他可以分配一个10分钟的时间盒，专门用于简化数据文件。在这个时间盒内，他可能获得了能够重现错误的最小文件，也可能没有进展。无论如何，在时间耗尽后，他停止调查，将那时的数据文件作为缺陷报告的附件提交。

原则上，测试人员应该提供“无冗余信息”的缺陷报告。但是，在紧张的测试进度下，测试人员只能用有限的时间去打磨测试信息。一般情况下，10分钟的时间盒能让测试人员付出足够的努力，获得恰当的进展。花费更多的时间很可能降低了团队效率，因为了解实现细节的程序员更适合调查复杂的错误。

当项目进度不紧张时，测试人员可以考虑为某个有趣的缺陷分配一个长度为30~60分钟的时

间盒，来彻底研究缺陷的来龙去脉。在这段时间内，他进行“刻意的练习”，通过调查该缺陷来学习产品的细节、测试工具的功能、调试技术的应用等。该时间盒的主要目的是通过一个真实的案例，来学习知识并练习技能。经过几次这样的练习，他可以在10分钟的时间盒内提供更多的信息。总体上，正常时间盒（短时间盒）是为了提供更好的缺陷报告，长时间盒是为了提高测试人员的知识与技能。

关于是否继续测试的问题，测试人员也可以采用时间盒来试探。他可以分配一个20分钟的时间盒对当前功能实施后续测试。当时间耗尽，他可以问自己如下问题。

- 这段时间的测试发现新信息了吗？
- 这些信息是否暗示还存在严重的缺陷？
- 继续测试的付出能获得好的回报吗？
- 为了尽快发现软件的严重错误，我是否应该测试其他尚未测试的功能？

通过思考这些问题，他可以更好地安排测试时间。其可能的选择包括再安排20分钟的时间盒来继续测试，或开始测试其他功能。

时间盒是一种常用的时间限制启发式方法 [Weinberg08]。它简单明了，又不失弹性。它使当前的任务获得稳定的时间，又确保其他有价值的任务不会被长时间延后。

2.2.2　通过技术调查发现更多的信息

缺陷提供了被测软件的新信息。利用新信息，测试人员可以对相关领域进行测试，从而发掘出更多的缺陷。做后续测试有两个好处：一是可能会发现一些额外的信息，使原先的缺陷报告更完整；二是可能会发现一些相似或相关的缺陷，使测试人员对软件的整体质量更了解。

后续测试的一个切入点是评估当前缺陷的风险。评估风险要考察两个因素：风险暴露的可能性和风险暴露的损失（更多讨论请参考8.3节）。

- 考察“风险暴露的可能性”是要评估风险转变为实际失败的概率。对于软件缺陷，测试人员需要估计用户遇到此缺陷的可能性，这通常要求测试人员发现暴露此缺陷的用户情景。对于复杂的软件，这样的用户情景可能有多个。测试人员的目标是找到最容易暴露缺陷的用户情景。
- 考察“风险暴露的损失”是要估计失败所造成的损失。对于软件缺陷，测试人员需要估计缺陷暴露所带来的最大用户损失。

为了发现这些信息，测试人员需要设计多种测试，以考察更多的情况。例如，某位测试人员测试Microsoft PowerPoint，该软件能够导入多种格式的图片，将它们放在页面上，并提供一组“图片特效”（如柔化边缘、添加阴影、添加倒影、调节亮度等）去美化图片。测试人员在测试打印

功能时，偶然发现页面上的一个图片在打印结果中丢失了。这是一个严重的缺陷，测试人员决定再进行一些测试，来调查问题的普遍性和严重性。

- 他用不同型号的打印机来打印有问题的文档，来检查缺陷是否与打印机硬件或驱动相关。
- 有些软件可以在操作系统中注册“虚拟打印机”，将文档发送到该打印机能够生成特定格式的文档（通常是PDF文档）。他将有问题的文档发送给“虚拟打印机”，并检查生成的文件。如果缺陷重现，说明错误与打印机类型和驱动无关。
- PowerPoint提供了二次开发功能，允许第三方开发者调用打印API（应用程序编程接口）来打印文档。他会调用该API，以检查错误是否影响到了第三方开发者。
- 他会删去文档中的一些文字和图片，再检查丢失的图片是否能打印。有时打印错误是其他可打印对象引起的，丢失的图片不一定是问题的根源。
- 他会替换不能正确打印的图片。如果该图片是BMP格式，他会将其替换为其他BMP格式的图片，再添加一些PNG格式的图片，然后打印。通过测试，他就能评估问题的根源是特殊的图片（如果其他BMP格式的图片可以正确打印），还是特定的图片格式（如果BMP格式的图片都不能打印，但是PNG格式的图片可以打印）。
- 他会检查图片是否被施加了一些“图片特效”。如果有特效，他会逐一去除“图片特效”并打印，以检查罪魁祸首是否为特定的“图片特效”。
- 如果发现“图片特效”是打印错误的原因之一，测试人员会联想到“艺术字”也有相似的“文字特效”（如柔化边缘、添加阴影、添加倒影等）。他会在文档中增加一些“艺术字”和相应的“文字特效”，并测试它们能否正确打印。
- 他回忆发现缺陷的过程，想起在打印之前，他对文档进行了许多操作。于是，他将文档复制到另一台测试机，启动被测的排版软件，再次打印该文档。通过该测试，他可以判断打印错误是否与排版软件的状态有关。注意，他并没有关闭最初发现错误的软件，因为该进程可能蕴含重要的错误信息。
- PowerPoint还可以生成PDF文件。考虑到生成PDF的代码与打印代码共享了一些代码，测试人员会使用有问题的文档来生成PDF文件。
- 在任务管理器中，测试人员发现软件在打印时会申请许多内存。是不是因为内存管理遇到错误，而不能正确打印呢？测试人员会切换到另一台内存更小的机器上，以检查在内存受限的情况下打印会遇到什么问题。通常，测试人员会准备两台（或多台）配置差异较大的测试机（如不同的CPU、内存、显卡、操作系统等），能够快速发现一些软硬件配置导致的问题。

有时后续测试并不需要覆盖以上所有变化，但是某些严重的缺陷可能要求更广泛的测试。为了启发思路，测试人员可以参考James Bach提出的7大产品元素，从而多角度地思考测试覆盖[Bach12]。

- **结构**：软件所拥有的组成元素。在文件级别，组成元素是构成软件的各种文件；在代

码级别，组成元素包括语句、函数、类等。从结构角度考虑测试覆盖，测试人员需要用测试去覆盖更多的结构元素。在上例中，测试人员所做的后续测试大多提高了结构覆盖。

- **功能**：软件所拥有的功能。为了获得更多的信息，测试人员需要测试相关的功能，并确保覆盖了每个功能的相关细节。在上例中，测试人员测试了相关功能“生成PDF文件”和“文字特效”，从而检查了缺陷的影响范围。
- **数据**：软件所使用的数据。由于程序能够接纳、处理、输出的数据是无穷尽的，测试人员需要将数据分类，并确保测试了每个分类的典型案例。在上例中，测试人员用多种方式提高了数据覆盖：测试新的BMP文件、测试PNG文件、测试艺术字、测试“图片特效”产生新的图片对象、测试“文字特效”产生新的文字对象、测试简化后的文档等。
- **接口**：软件所提供的操作界面，如用户界面、系统界面、API、编程平台的SDK、数据导入和导出功能等。在上例中，测试人员覆盖了软件提供的API，还通过导入更多的图片覆盖了图片导入功能。
- **平台**：软件所依赖的软硬件环境，这包括硬件平台、网络环境、操作系统、软件依赖的其他软件和网络服务等。在上例中，测试人员用新的打印机和新的测试机提高了平台覆盖。
- **操作**：软件可能的使用方式。测试人员将功能组合成流程，用不同的顺序调用功能，以发掘用户操作时可能遇到的问题。在上例中，测试人员引入了一些新的操作流程：增删图片和文字后再打印、启动新的被测进程后再打印。
- **时间**：软件与时间相关的元素。许多错误与时间或时序有关，例如多线程死锁、多线程竞态条件、客户端与服务端时间不同步引发的冲突等。在并发编程和网络编程日趋重要的今天，这些问题需要引起测试人员的重视。上例的缺陷与时间没有明显的关系，所以没有专门的后续测试，不过一些测试用例无形中提高了时间覆盖。例如，用新进程打开文档后直接打印和长时间编辑文档后再打印，是两种不同的操作序列，它们可能导致软件用不同的方式去处理图片缓存，进而导致不同的行为。

测试覆盖并非唯一的测试切入点，测试人员还可以利用漫游测试（参见5.4节）、基于典型缺陷的快速测试（参见5.5节）等方法来测试产品。其目的是通过多样化的测试来获取更多信息，从而更好地评估缺陷的风险。如果打印丢失只出现在某些图片和某个特效的组合上，那么问题的普遍性较低，但严重性较高（因为缺陷导致用户数据丢失）。如果打印丢失出现在所有的BMP图片上，或相似的情况也出现在文字特效上，或生成的PDF文件也丢失图片，那么问题的普遍性和严重性都较高。

通过测试，测试人员可能发现了新的缺陷，可能发现了原缺陷的新线索，也可能没有太多收获。无论如何，他都可以在缺陷报告中记录做了什么和得到了什么结果，这些都为缺陷的评审和修复提供了有力的支持。

2.2.3 处理难以重现的缺陷

难以重现的缺陷是测试人员最常遇到的工作挑战之一。以悲观的角度来看，难以重现的缺陷意味着软件的行为受到未知因素的影响，搞清楚这些因素即便不是“不可能的任务”，也需要大量的时间。以乐观的角度来看，它既然出现过，就有可能重现；即便不能修复，项目团队也知晓它的症状和风险。很多时候，重现一个缺陷确实需要一些运气，但是测试人员不能单纯地依赖运气，他需要从多个方面努力应对挑战。

第一，测试人员需要以正确的态度面对缺陷。测试人员的字典中没有“不能重现的缺陷”，他会说这个“间歇性缺陷”很难重现，我尚未掌握稳定重现它的所有因素。这种想法使测试人员能够以积极的态度来面对困难。具体而言，他应该具有如下信念。

- 缺陷是可以重现的。
- 目前，我已经掌握了重现该缺陷的部分因素，利用它们尚不能重现该缺陷，或只能以较低的概率重现该缺陷。
- 我的任务是发现更多的因素，以稳定地重现该缺陷，或以更高的概率重现该缺陷。

第二，在测试过程中，测试人员应该为潜在的缺陷做好准备，当一个缺陷暴露时，他可以收集尽可能多的信息。在我的经验中，有些缺陷之所以没有重现，是因为我在第一次发现时没有记录足够的信息。原以为我能通过重现去记录更多的信息，却发现该缺陷“神秘地”消失了。无奈之下，我只好提交一份信息较少的缺陷报告。因为缺少线索，接到报告的程序员也没有重现该缺陷。最后，缺陷被解决为“没有重现”。

在吸取了教训之后，我会在测试之前和测试过程中做以下准备工作。

- 在测试机上安装调试器Windbg，并将它设为默认的事后调试器。当被测试软件崩溃时，Windows将启动Windbg来调试该崩溃[①]。
- 在测试机上安装Web调试代理Fiddler[②]。如果软件会用HTTP协议与其他计算机通信，我会在测试之前打开Fiddler，让它监控并记录软件的HTTP请求与响应。
- 在测试机上安装Process Explorer[③]。在测试之前启动Process Explorer，让它监视软件所使用的资源，重点关注CPU、内存、句柄等关键资源的使用情况。
- 在测试机上安装一款性能分析工具[④]。如果软件遇到性能问题，我会用它获取软件的性能跟踪数据。

---

① 关于Windbg的使用，可以参考熊力所著《Windows用户态程序高效排错》，全文下载：http://www.cnblogs.com/lixiong/archive/2010/02/11/1667516.html。

② 详见http://www.fiddler2.com/fiddler2/。

③ 详见http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx。

④ 在Windows 8上，Windows Performance Toolkit是一款强大的性能分析工具，它被包含在Windows 8 SDK中，下载地址：http://msdn.microsoft.com/en-us/windows/desktop/hh852363.aspx。

不难看出，这些监控工具都属于调试诊断工具。因为它们依赖于一些底层的高级技术，所以学习和使用难度较高。不过，掌握它们的基本功能就可以采集到许多有价值的信息。例如，内存转储文件记录了一个进程在某个时刻的内存状态，对于某些故障（例如进程崩溃）的诊断很有帮助。在提交缺陷报告时，测试人员可以考虑将内存转储文件作为报告的附件，以便程序员调试。生成内存转储文件有多种方法，本节介绍3种最常用的方法。

图2-1展示了如何利用任务管理器来生成指定进程的内存转储文件。测试人员选中目标进程后，右击鼠标，在弹出菜单中点击“创建转储文件”，就可以获得该进程的内存转储文件。该方法只使用Windows操作系统自带的工具，适用于所有安装Windows系统的测试机。

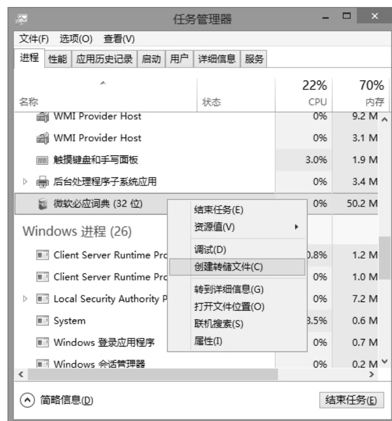


图2-1　使用任务管理器生成内存转储文件

图2-2展示了如何利用Process Explorerer来生成指定进程的内存转储文件。测试人员选中目标进程后，右击鼠标，在弹出菜单中点击“生成转储”（Create Dump），再选择“生成完整转储”（Create Full Dump）就可以获得该进程的内存转储文件。该方法允许测试人员提供转储文件的保存路径和文件名，在实际测试中较第一种方法更为方便。

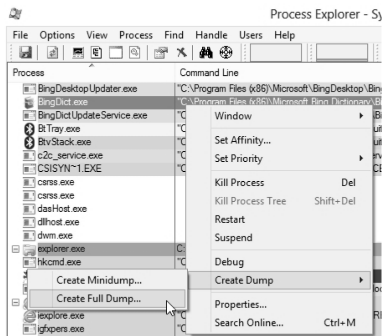


图2-2　使用Process Explorer生成内存转储文件

图2-3展示了一个注册表文件crash_auto_dump.reg，双击它可以将其记录的设置导入Windows系统的注册表。根据该设置，Windows系统会对崩溃的进程调用命令："c: \\debuggers\\windbg.exe" -p %ld -c ".dump /ma /u c: \\dump\\crash.dmp;q\"。该命令调用安装在c:\debuggers目录的windbg.exe（测试人员需要先将Windbg安装在该目录下）将崩溃进程的内存转储文件写入c:\dump目录。这样，测试人员可以自动获得崩溃进程的转储文件，使得一些罕见的崩溃错误能够获得有益的诊断信息。

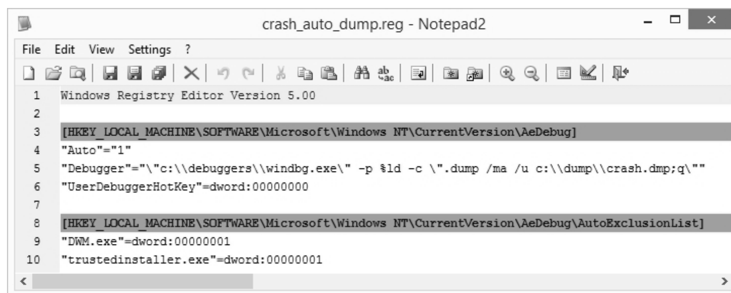


图2-3　使用Windbg自动生成崩溃进程的转储文件

本节介绍的工具适合Windows平台上的软件测试。在其他操作系统和运行环境上工作的测试人员可以结合具体任务，采用其他调试诊断工具，以便第一时间捕获缺陷的信息。熟悉这些工具，并让它们随时可用，能显著提高测试效率。

第三，如果第一次收集的信息并不能重现或定位缺陷，测试人员需要做后续测试[①]。测试之前，测试人员需要增强软件运行的可见性。缺陷没有重现说明还有一些测试人员没有观察到的关键因素，如果不提高可见性，很可能还是观察不到这些因素。因为软件运行非常复杂，测试人员需要熟悉软件的运行机制和调试工具，才能有效地观察。以下是一些常见的观察方法。

- **软件错误可能是因为软件不能获得某种操作系统资源，如内存、文件、网络端口、注册表键值等，也可能是因为它错误地使用了资源**。测试人员需要用工具（如Process Explorer）观察软件和操作系统使用和持有哪些资源。
- **软件错误可能是因为软件处于不正确的状态**。这通常体现为某些变量的值出了问题。测试人员可以用调试器（如Windbg）观察相关变量的值，并留意它们的变化。
- **软件错误可能是因为操作的顺序或时序不正确**。测试人员可以研究软件的日志，了解软件的每一步操作及其耗时。
- **软件错误可能是因为软件所调用的软件或服务返回了它不能处理的数据**。如果软件调用网络服务，测试人员可以用工具（如Fiddler）监控网络通信，以了解软件发送了什么数据、收到什么数据。如果所调用的软件或服务有日志，测试人员需要将日志记录设置为最详细的级别，并仔细阅读日志内容。一些诊断工具（如Windows Performance Toolkit）能够

① James Bach的文章How to Investigate Intermittent Problems（http://www.satisfice.com/blog/archives/34）提供了许多好的测试想法。

观察进程之间的调用关系，有助于识别软件之间的依赖关系。

- **软件错误可能是数据库返回了它不能处理的数据**。测试人员可以浏览数据库中的数据，检查返回数据的表、视图、存储过程、函数等对象，以推测数据的可能取值。
- **有些软件内建了一些调试辅助功能，能够提供软件运行的详细信息**。测试人员可以启动这些功能或安装具有这些功能的调试版本。

第四，测试人员应该用“科学实验”的策略来指导缺陷重现：提出假设，根据假设设计实验，用实验结果推翻假设或提出新假设。在观察的基础上，测试人员结合自身经验，可以提出一些对错误原因的猜测。然后，他选择一个猜测，在它的指引下，设计一批测试去尝试重现缺陷。如果这些测试没有获得有价值的新发现，他可以选择另一个猜测，再设计并执行测试。如果测试中发现了新情况，他可以提出新猜测并记录下来，然后根据新猜测开始测试。

重现缺陷很容易陷入盲目的尝试，直到测试人员感到疲倦而放弃，也没有多少进展。提出错误猜测使得测试有明确的目标，在它的指引下测试人员可以自由发挥，而不会迷失方向。整个过程的特征是设定明确的目标、用实验去获得信息、根据新信息建立新目标。

第五，调查一个难以重现的缺陷与软件调试非常相似，测试人员可以借鉴一些调试原则和方法。调试专家David J. Agans的9条调试规则 [Agans06] 不但可以指导调试，还能帮助测试。

- **理解系统**（Understand the system）。解决一个具体问题，很多时候需要在整体上理解系统。重现一个缺陷也要求测试人员在宏观上和细节上都能把握软件。在项目全程，测试人员需要积极地学习被测软件的领域知识、产品架构、代码实现、技术平台、调试工具等。只有较全面地理解软件，才能有效地分析信息、设计实验、快速测试。软件开发与测试没有捷径，持续积累才是正途。
- **制造失败**（Make it fail）。影响软件行为的因素包括软件自身的状态和软件的外部输入（用户的输入、操作系统的输入、其他软件的输入等）。测试人员应该列举这些影响软件行为的因素，并分析哪些因素可能与当前缺陷有关。然后，他需要大量尝试这些因素的取值组合，并试着总结缺陷重现的模式。有时测试人员需要求助于程序员，让他提供更好的测试钩子以控制软件的状态和输入，或提供更好的调试日志以了解软件的状态和输出。
- **观察先于思考**（Quit thinking and look）。在猜测引发错误的原因之前，应该仔细地观察软件的状态、操作系统的状态、所依赖的软件或服务的状态，并分析软件的输入数据和输出数据。细心观察可以更好地推测缺陷原因，设计更有效的测试，相反，凭空猜测很可能使测试误入歧途，费时费力却没有收获。
- **分而治之**（Divide and conquer）。如果运行一次测试需要大量的准备工作（如初始化数据库、复制数据文件、重启服务等），测试就不能提供快速的反馈，缺陷调查的效率就很低。为此，测试人员需要建立一个可以快速测试的环境，这通常需要隔离一些无关的数据、软件和服务，也需要测试钩子的帮助。在此环境中，测试人员通过逐次逼近来缩小搜索的范围，即通过测试来排除一些影响软件的因素，从而慢慢锁定若干重要的因素。

- **一次只做一处修改**（Change one thing at a time）。在科学实验中，科学家一次只修改一个变量的值，从而了解该变量对于实验结果的影响。同样，在测试中，测试人员一次只修改一个变量的值，从而了解它对软件行为的影响。
- **保持审计跟踪**（Keep an audit trail）。在调查过程中，要记录下所做的测试和测试结果。魔鬼隐藏在细节中，而人的记忆总是不可靠的。应该将测试、软件和环境的细节记录下来，作为缺陷报告的一部分。
- **检查基本假设**（Check the plug）。如果测试人员的调查没有进展，他也许要质疑一下他的基本假设。面对复杂的问题，测试人员的思考常常基于一些能简化情况的假设，例如他会假定“服务器返回的数据应该是没问题的”、“这个模块一直很稳定，不会出错”、“错误检查代码一定能过滤掉无效的输入数据”等。这些假设在大多数时候成立，但是并非永远如此，要质疑并检查它们。
- **获得全新视角**（Get a fresh view）。如果重现的努力没有成功，测试人员可以向程序员或测试同伴请教。软件是如此的复杂，每个人都有盲点。向别人解释问题，会强迫测试人员组织思路，这有助于他跳出原来的思维模式，获得新的见解。也许叙述完毕，测试人员就获得了新的测试灵感。此外，被咨询人可能知道一些知识，能够提供有效的测试策略或关键线索，这会显著提高调查效率。
- **如果你没有修复它，它就一直存在**（If you didn't fix it, It ain't fixed）。当重现缺陷的努力失败时，测试人员不能简单地说：“它一定随风而去了。”缺陷在测试环境中不出现，但可能在用户的环境中出现，更可能在公司领导向媒体做公开演示时出现。为了让项目团队获得必要的信息，测试人员应该提交缺陷报告。 调查不可重现的缺陷可能是一个“时间黑洞”。测试人员应该根据缺陷的严重程度，设定不同长度的时间盒。在时间耗尽时，他可以自问：“是否追加一个时间盒继续调查？在下一个时间盒中能获得突破吗？对于项目而言，是继续调查还是开始新的测试更有利？”经过评估，他可以决定是继续调查，还是结束调查并提交缺陷报告。

第六，在结束调查后，测试人员需要提交缺陷报告。如果测试人员还是未能稳定重现缺陷，他需要非常仔细地编写缺陷报告。因为信息不足，草率的缺陷报告大多被解决为“不能重现”或“不予修复”。为了让程序员和缺陷评审者获得足够的信息，测试人员需要在缺陷报告中写明以下内容。

- 在报告的开头坦承该缺陷不能稳定地重现。
- 估算该缺陷的重现概率。它只出现了一次吗？还是在某个情景中，以N%的概率重现？对概率的估算不必追求精确，只要根据测试记录进行简单计算即可。
- 该缺陷可能重现的情景有哪些？在各个情景中重现的概率是多少？
- 如果该缺陷重现，所造成的最严重后果是什么？
- 为了重现该缺陷，做了哪些测试，测试结果如何，在测试中发现了什么？
- 是否有人提交过类似的缺陷？那些缺陷提供了哪些信息？

- 根据现有信息，缺陷人员对错误根源有哪些推测。只要合情合理，有事实依据，测试人员就可以大胆地发表自己的看法。

随着项目的发展，测试人员可以分析已知的难以重现的缺陷。从这些缺陷中，他也许可以总结出若干模式：某些缺陷有相似的调用栈、某些缺陷在特定的环境中可能重现、某些缺陷在特定的操作序列下可能出现等。与程序员或测试同伴讨论这些模式可能会发现缺陷的根源，也可能会获得重现缺陷的新思路。

2.3　编写高质量的缺陷报告

本节介绍一些技巧和实践方法，来帮助测试人员编写高质量的缺陷报告。因为不同的项目团队对测试人员有不同的期望，读者在采纳这些基本实践时，需要做一些因地制宜的变化。

本节假设项目团队使用基于数据库的缺陷管理系统来跟踪缺陷报告。团队成员可以向系统提交新的缺陷报告，修改和解决已有缺陷报告，查询符合一定条件的缺陷报告。在此过程中，缺陷管理系统提供审计跟踪功能，记录对缺陷报告所做的一切修改。

2.3.1　为每一个缺陷单独提交一份缺陷报告，小缺陷也是如此

为了提供完整的信息，测试人员应该报告他发现的所有缺陷。为了让每一个缺陷都得到审计跟踪，测试人员应该为其提供独立的缺陷报告。

有时，测试人员会在一份报告中提交了几个缺陷。这可能会导致一些问题。

- 缺陷报告的数目会少于缺陷的数目，这使缺陷统计不能获得准确的信息。
- 缺陷报告中的缺陷可能有不同的根源，需要不同的解决方案，需要不同的程序员（甚至不同的项目团队）来解决。一份报告不能很好地跟踪所有缺陷的修复。
- 一份缺陷报告只有一个优先级和严重性，但是它所提交的缺陷可能拥有不同的优先级和严重性。这使得缺陷报告不能传递准确的优先级和严重性。
- 因为不能很好地标识与跟踪每一个缺陷，某个缺陷可能被程序员和测试人员所忽略，以致于没有被修复。

为了避免这些问题，测试人员需要将此类缺陷报告拆分成多个缺陷报告，让每一个缺陷得到独立的报告。

然而，缺陷管理系统的关键问题是行政性的，而不是技术性的[Kaner99]。在某些项目环境中，测试人员提交缺陷报告的动力会被压抑。一个典型的情况是项目管理者用缺陷报告的数量来评价程序员的绩效，这势必导致程序员对缺陷报告很敏感。他也许会对测试人员说："你如果发现缺陷，请不要提交缺陷报告，而是发邮件通知我。"这让测试人员处于两难的境地：一方面，他知

道向项目团队提供高质量的信息是测试人员的基本责任，隐藏信息将伤害测试人员的“信誉”；另一方面，他不想破坏与开发伙伴的关系，这是高效测试的基本条件之一。

坦率地说，测试人员没有能力和资源去解决“不良管理”所带来的问题，这是项目经理或更高层经理的职责。面对此类情况，测试人员还是应该正常报告每一个缺陷，因为隐藏缺陷将伤害项目团队对测试人员的信赖，而“可信赖”是一个信息提供者的工作基础。同时，他需要向“受影响”的程序员解释他的做法，并强调缺陷报告从来都是对事不对人。更积极的行动是，他应该当面向他的领导（通常是测试经理或项目经理）解释当前的管理流程给测试工作带来了困难，并将对项目产生不好的影响。如果能联合其他测试同事一起提出建议，领导者会更容易看到问题之所在。

2.3.2　仔细编写缺陷报告的标题

在阅读缺陷报告时，读者总是先读到标题。例如，缺陷评审小组会查询所有活跃的缺陷，查询结果是一个缺陷标题的列表，缺陷标题为进一步的分拣提供了基础。再例如，程序员会查询所有指派给自己的缺陷，通过阅读缺陷列表，来了解当前待修复的缺陷。再例如，测试经理报告测试进展，其报告会列举当前活跃的缺陷（体现为一个缺陷标题的列表）来描述软件开发的状态，报告的读者对缺陷所有判断皆来自标题。

为了帮助缺陷报告的读者更快、更准确地理解缺陷，测试人员需要认真编写标题。一个较好的模式是“条件–失败”[Karen08]，即阐述在何种情况下软件会发生什么样的失败。以下是一些具体的例子。

- 当软件不能申请到足够的内存时，打印结果将丢失页面上的一些图片。
- 当打开一个旧版本的文件（版本号：2.0）时，软件报告打开文件失败（错误代码：XYZ）。
- 当程序退出时，它可能引发间歇性崩溃（崩溃时所调用的函数是ABC）。

标题要客观陈述软件所遇到的问题，描述触发问题的情景和问题的症状。这有助于读到标题的人推断缺陷的严重性和可能的技术原因。测试人员还可以在标题中加入一些技术细节，如错误代码和函数名，这使得相似症状的问题可以相互区别。

2.3.3　像编写详细测试用例那样编写重现步骤

在编写缺陷的重现步骤时，测试人员可以参考详细测试用例的编写规则，使得重现步骤尽可能地具体、无歧义。

- 如果缺陷与环境有关，在重现步骤之前增加“测试配置”一节，写下测试所使用的操作系统、网络服务、网页浏览器等信息。

- 在重现步骤中，为每一个操作编号，以清楚地表明第1步做什么、第2步做什么。
- 不要假定缺陷报告的读者对软件很熟悉，要清楚地写下每一步操作的内容，例如点击了什么按钮、在哪个控件中输入了什么字符等。
- 不要包含不必要的步骤。
- 清楚地写下执行操作序列后的“预期结果”。
- 清楚地写下执行操作序列后的“实际结果”。
- 如果可以提供更多的信息，可以将它们写入“附加信息”一节。

以下是一个重现步骤的例子。

测试配置：测试机安装64位Windows 8.1操作系统，32GB内存。

重现步骤：

(1) 关闭计算机上的所有窗口。
(2) 启动产品。
(3) 用产品打开缺陷报告附件中的文档。
(4) 点击“文件 → 打印”。
(5) 在打印对话框中选择“Micorosft XPS Document Writer”。
(6) 点击“打印”。
(7) 保存XPS文件。

预期结果：所生成的XPS文档共10页，每页正确显示原文档的内容。

实际结果：所生成的XPS文档共10页，但是第8、9、10页上缺少原文档所包含的图片。

附加信息：在打印过程中，产品进程的内存占用持续增长，从200MB一直增长到1.1GB。在打印结束后，内存占用降回400MB。详细信息请参考附件中的“内存占用变化趋势.jpg”。

2.3.4 使用缺陷模板来提交缺陷

许多缺陷管理系统支持缺陷报告模板，测试人员应该充分利用该功能。首先，他可以利用模板为一些缺陷报告字段设置默认值，这提高了编写报告的速度。其次，缺陷模板定义了缺陷报告的结构，能提醒测试人员报告一些重要的信息。以下是一份简化了的缺陷模板，它用于提交“打印”相关的错误。

标题：[打印] 在什么情况下，打印发生何种故障

功能路径：产品家族X\产品Y\打印

严重程度：严重

测试配置：请提供操作系统、打印机型号等信息

重现步骤：

(1)
(2)
(3)

预期结果：

实际结果：

附加信息：

请考虑提供如下信息。

- 用户影响
- 重现此缺陷的其他情景
- 有助于调试的任何信息（如屏幕截图、函数调用栈、运行日志、内存转储等）
- 该缺陷能否稳定重现？如果是间歇性重现，重现概率是多少？有没有更多的信息？
- 相关缺陷

这份缺陷模板包含“用户影响”等提示词，它们提醒测试人员从特定的角度来提供信息。测试人员可以根据项目的特点，在缺陷模板中加入恰当的提示词。在报告某个缺陷时，测试人员可能会忽略一些提示词，因为它们并不适用于当前缺陷，但这是测试人员思考后的决定，而不是遗忘此类信息而导致的疏漏。

在日常工作中，我会制作几份缺陷模板，分别针对不同的功能模块或缺陷类型。随着项目的进展，我会修订缺陷模板或增加新的模板，使得它们符合项目团队对缺陷的要求。

2.3.5 在编写缺陷报告时，要考虑缺陷查询

有时，测试人员需要查询缺陷管理系统。例如，他发现了一个缺陷，在提交缺陷之前，他会搜索缺陷管理系统，以了解该缺陷是否已经被提交。又例如，在发现一个缺陷后，他想起以前曾经提交过一个相似的缺陷，为了了解那个缺陷是如何解决的，他会查询缺陷管理系统。再例如，有人询问测试人员为什么软件是如此设计的，他回忆起是某个缺陷修复决定了当前的设计，于是他会搜索那个缺陷。

可见，查询缺陷是一件常见的任务。为了使查询更轻松，测试人员在编写缺陷报告时要略加斟酌。在编写缺陷标题时，他可以自问：“如果几周后我查询此报告，我会使用什么关键词来搜索标题？其他测试人员会使用什么关键词？”在列举了一组关键词之后，他可以选择一两个写入

标题。如果缺陷报告提供了“关键词”字段，他可以将关键词们一起写入该字段。如果没有专属的“关键字”字段，测试人员可以将这些关键词写入报告正文。

有时，测试人员会搜索“重现步骤”或“附加信息”。为此，测试人员应该将尽可能多的细节写入这些部分。最典型的例子是软件崩溃时的函数调用栈。如果缺陷报告总是包含崩溃时的函数调用栈，测试人员用函数名搜索就可以发现此函数有关的软件崩溃。

2.3.6　链接相关的缺陷

有些缺陷出现在常见场景或共享模块中，会被多人发现。为了避免重复提交某个缺陷，测试人员需要在缺陷管理系统中搜索相似的缺陷。如果该缺陷已经被提交，测试人员可以将自己发现的信息补充到已有的缺陷报告中。测试人员不应该假设很明显的缺陷一定会被同事提交，根据查询结果采取行动才是合理的策略。

有时，测试人员会发现一些症状相似的缺陷，但并不能确认它们的错误原因是相同的。对于这种情况，他应该为每一个缺陷单独提交一个缺陷报告，并将它们链接在一起，从而让程序员或缺陷评审者了解完整的信息。有些缺陷管理系统支持“链接”缺陷，只要输入相关缺陷的编号，系统就可以将几个缺陷链接在一起。如果缺陷管理系统不支持“链接”，测试人员可以选择一个缺陷作为“主缺陷”，在其他缺陷报告中写下主缺陷的编号，并在主缺陷中记录其他缺陷的编号。

2.3.7　注意缺陷报告的可读性

缺陷报告是面向整个团队的公开文档，应该提高可读性以服务读者。测试人员不必反复推敲报告的字句，他只需要注意一些基本规则，就可以提高报告的可读性。

- **避免错别字和病句**。如果用英语撰写报告，利用字处理软件进行拼写和语法检查，能够立即发现并修正明显的问题。对于中文报告，写完之后通读一遍就可以改正大多数文字错误。
- **尽量用数字列表来记录重现步骤**。
- **尽量用符号列表来罗列事实**。
- **用空行分隔报告的不同部分，在视觉上突显报告的结构**。
- **用屏幕截图、视频录像、内存转储等附件来提供更多的信息**。有些缺陷用文字很难描述，但是一幅截图或一段录像就可以很好地说明问题。关于视频录像工具，智能手机是一个很好的选择。它适用于任何软件和操作系统，且对被测软件的运行没有任何影响。

2.3.8　客观中立地书写缺陷报告

缺陷报告是对缺陷的客观陈述，只传递事实，不评价任何人的工作。客观的风格使得测试报

告更加可信，倾向性明显的报告则让读者怀疑测试人员的专业态度，反而不利于缺陷的修复。

- 测试人员需要了解项目团队对缺陷严重程度和优先级的定义。坚持按照团队约定来填写这些字段，不但能够保证缺陷报告的一致性，还可以避免一些不必要的争执。
- 有时，测试人员要对缺陷的用户影响或可能原因做一些评论，他应该先列举事实，然后再进行谨慎的推断，并写清楚推理过程的因果逻辑。
- 缺陷报告只报告错误，不要评价软件设计，也不建议解决方案。测试人员可以当面与产品经理、程序员等讨论软件设计，询问实现细节，了解对方观点，并发表自己的意见。

2.4　对不予修复的缺陷进行上诉

有时，缺陷报告会被解决为“不予修复”或“延迟修复”。如果测试人员不同意这样的决定，他需要立即进行上诉。在许多项目中，上诉是将缺陷报告提交给缺陷评审小组，请他们再次审查缺陷报告，然后作出最终的决定。

在上诉之前，测试人员需要做一些准备工作。

第一，他可以再做一些测试，以研究缺陷的普遍性和严重性。目的是用证据说明缺陷可以在主流场景中暴露，或缺陷能带来严重的危害。

第二，他要改进缺陷报告，以强调缺陷的普遍性或严重性。除了加入后续测试的结果，他还可以请教专家（资深员工、领域专家、用户代表等），将他们的意见写入缺陷报告。此外，他还可以收集一些资料，来证明缺陷值得修复。例如，某个打印缺陷只在特定型号的打印机上出现。他可以利用互联网搜索市场报告，以了解该型号的打印机一共销售了多少台、市场前景如何、其典型用户具有什么特征。如果发现这是一款畅销的打印机、其用户与被测软件的用户有很大的交集，那么将这些信息写入缺陷报告会显著提高其说服力。

第三，他可以与缺陷评审小组的成员做面对面的交流。当面陈述问题能够更快速地交换意见，更好地说明问题。特别是测试人员可以当面演示缺陷的重现情景，从而更生动地说明缺陷对用户的伤害。另一方面，他可以了解缺陷评审小组的意见，知晓他们需要什么样的证据才会作出修复的决定。此外，缺陷评审者往往是团队的管理者或业务专家，他们会从项目进度、价值取舍、商业影响等多个方面评审缺陷。与他们交流，测试人员能够更好地理解业务和项目，更好地理解其决定的根本原因。

缺陷上诉的本质是说服别人采纳自己的建议。在收集证据和说服的过程中，测试人员的“人脉”会有潜移默化的作用。这里的人脉是指测试人员的信誉和团队对其能力的认可程度。对于人脉好的测试人员，团队成员会认为他的意见是值得参考的，也愿意力所能及地帮助他。

测试人员建立人脉的主要方法有两条。

- **通过持续提交高质量的缺陷报告来建立信誉**。在许多项目团队中，缺陷报告是测试人员的主要工作成果，程序员和缺陷评审者通过它们来评价测试人员的专业水平。文如其人（You are what you write）[Kaner01]，高质量的缺陷报告让测试人员获得信任，松散的缺陷报告让测试人员失去尊重。从这个角度，赢得某次的缺陷上诉是长期努力的结果。
- **通过帮助团队成员，让他人认可他的能力和对团队的贡献**。除了报告缺陷，测试人员还通过其他途径来服务团队，例如测试程序员的私有版本以提供快速反馈、深入调查一个严重的缺陷、周密地测试一个即将上线的补丁等。在开发过程中，许多任务之间并没有很清晰的界限，如发现和查明缺陷[Weinberg08]。测试人员不要斤斤计较或争论特定的活动是否属于测试，倘若有利于团队，就应该投入时间去做。这不但能获得他人的肯定，还是很好的能力锻炼。

例如，有一次我发现了一个由UI线程（用户界面线程）引发的死锁：UI线程将一个任务交给线程池的一个工作线程，工作线程为了完成任务要申请一个锁对象，该锁对象被UI线程持有，而UI线程正在等待工作线程完成任务，于是这两个线程发生死锁。当时，此失败是一个棘手的问题：一方面，错误发生时，UI线程仍旧能够处理用户输入，所以看上去它没有死锁；另一方面，这是一个间隙性的失败，在开发者的机器上不能重现，在我的测试机上只能够偶尔重现。为了避免该缺陷被解决为“不予修复”或“不能重现”，我花了一个多小时调查该错误，终于重现死锁，并用调试器找到了引起死锁的工作线程和锁对象。我提交了这些信息后，开发者立即表示我的报告很有价值，并迅速修复了这个提交了多日的“悬案”。如今回想，我本可以更快地发现错误根源，但是我对代码不了解，调查时走了一些弯路。不过，通过这次调查，我较完整地理解了这部分代码，今后我能够更有效、更自信地测试。还有一点感悟是，我和程序员非常幸运，因为我的测试机能够“偶尔”重现该死锁。如果该缺陷遗漏到用户环境再发现，我们可能需要用几倍的时间来调查和修复。

2.5　周密地测试缺陷修复

当缺陷被修复后，测试人员需要测试软件的新版本，以检查缺陷是否得到修复、修复是否引入新的缺陷。该活动体现出软件测试的迭代本质：测试人员总是循环测试软件的版本序列，后续的测试任务基于先前发现的信息（如缺陷报告），而先前测试获得的知识会改进后续测试的测试策略。

测试缺陷修复大致可以分成两类。第一类是测试普通影响力的缺陷修复，例如测试项目中发现的普通缺陷；第二类是测试高影响力的缺陷修复，例如测试架构级别的改动、测试发布前夕做的缺陷修复、测试已发布产品的补丁等。第一类任务只要求做小范围的回归测试，工作量较小。第二类任务要求测试人员对某个功能和情景做周密的测试，工作量大，可能需要一个或多个工作日才能完成。

对于普通影响力的缺陷修复，测试人员需要尽快完成测试，以提供及时的反馈。程序员刚提交代码时，能够回忆起代码细节，如果测试发现新问题，他可以快速地修正。此外，有一些缺陷修复更改了软件的设计，这可能是程序员、测试人员和程序经理讨论的结果。如果测试发现新的设计仍旧存在问题，他们可以有时间再做设计。如果测试人员“积压”了一批待测试的缺陷修复，直到项目后期才测试，那么在测试中发现的问题很可能没有时间去修复。对于缺陷而言，其报告时机总是越早越好。

在开始测试时，测试人员可以将缺陷报告的重现步骤作为脚本测试用例，通过执行该用例来做基本检查。如果软件通过基本检查，测试人员需要再做测试来检查修复确实有效。以下是一些可能的测试策略。

- 测试人员可以查看代码的变更集，了解当前设计，从而更有针对性地测试。有时，测试人员并不能完全读懂代码细节，但是了解大致的设计，就能为测试提供有益的指导。如果必要，测试人员可以请教程序员，以深入了解设计。
- 测试人员可以考虑产品元素的7个分类，在结构、功能、数据、接口、平台、操作、时间上设计新的测试用例。不要完全复用旧的测试用例，应该设计新的测试用例来拓展测试的广度和深度。
- 测试人员可以对被修改的代码做“影响分析”（参见7.1节）。通过考虑被修改代码的输入、输出、副作用，他可以更好地设计测试。例如，他可以思考：被修改的代码从何处获得数据？上游代码会不会产生它不能处理的数据？软件会不会用反常的方式调用它？被修改的代码会调用什么代码？这些调用会不会产生出人意料的结果？被修改的代码会产生什么数据？下游代码能否妥当地处理这些数据？被修改的代码会不会修改全局状态？被修改的全局状态会不会产生不良的影响？
- 与“影响分析”相似，测试人员可以考虑哪些功能与被修改的功能集成或共享数据。他可以测试这些相关功能，以检查新的设计是否破坏已有的功能。
- 测试人员可以针对缺陷类型做一些有针对性的测试。例如，为了提高程序的响应速度，程序员将一些计算转移到工作线程中。对此，测试人员可以制造一些情景，使工作线程的计算抛出异常，以测试工作线程能否妥当地处理可能的异常。有时程序员会漏掉一些异常处理，而未被处理的异常很可能导致程序崩溃。

对于高影响力的缺陷修复，测试人员应该细致地测试新设计。考虑到测试时间较长，他可以将其视为一个微型测试项目进行管理。在此过程中，他根据产品的成熟度确定测试策略[Kaner01]，并动态调整测试活动。

例如，测试人员接到一个为期一周的测试任务：测试一个即将发布的产品补丁。他可以考虑按如下方式组织测试。

- 周一，他安装补丁，并执行冒烟测试，目的是在第一时间发现严重的缺陷，如补丁无法安装、基本操作执行失败、基本情景被阻塞等。然后，他会搜集补丁相关的信息，例如缺陷描述、修复说明、被修复功能的规格说明和测试文档、互联网上相关的业务或技术文档等。他一边积极地学习信息，一边“**同情地测试**”。测试的目的是检验信息的有效性，获得真实的反馈，并建立产品的测试模型，为进一步的测试做好准备。
- 周一下午，他会根据研究成果，编写一份1～2页的测试计划，列出已经执行和即将执行的测试想法，并发送给补丁程序员、资深同事、测试经理和产品经理等，邀请他们对测试计划提供反馈。下班时，测试人员会对未来的测试充满信心，因为他已经验证了基本的功能，并根据多样的信息源和真实的测试经验制订了测试计划。
- 周二，测试人员会“**积极地测试**”，对补丁和被影响的相关功能执行深入的测试。在测试的间隙，他会将新的测试想法写入测试计划，使测试计划反映他对产品的最新理解。
- 周三，测试人员已经对补丁和产品有了深入的理解，他也陆续收到同事对测试计划的反馈。他会再次综合各方面信息，完善产品的测算模型，从而“**多样地测试**”。他可以执行同事补充的测试想法，检查自动化测试的结果，利用测试工具执行压力测试，利用调试器检查产品代码的行为——他会运用一切手段去获取有价值的信息，直至新的信息不再出现。
- 周四下午，测试人员会将测试计划转化为测试报告。这不是单纯的文档工作，而是再一次审视测试情况，梳理测试思路，以查漏补缺，为最后一天的测试做好准备。如果测试情况显示测试人员需要更多的时间去完成测试，他可以立即向测试经理说明情况，并申请更多的测试时间。
- 周五，测试人员也许会收到本周所提交缺陷的修复，他会“**严谨地测试**”。在发布前夕，对任何一行代码或配置文件的修改，都要做细致的、一丝不苟的测试。测试内容往往是针对小范围代码的全面测试，包括简单的验证，也包括复杂的攻击。测试人员不会因为一周紧张的工作就放松懈怠，他会理解此时细小的错误将导致巨大的损失。
- 最后，他会提交测试报告，在客观描述的基础上，给出他对补丁的评价和建议。虽然测试时间结束了，但是测试可能并未结束。如果测试报告显示进一步的测试能够发现有价值的信息、能够揭示潜在的风险，测试人员应该与管理人员交流，建议进一步测试。管理层会根据市场、进度、技术等信息作出决策。

2.6　坚持阅读缺陷报告

在项目过程中，测试人员应该持续阅读他人提交的缺陷报告，从而更好地理解项目进展，并激发测试灵感。

许多缺陷管理系统支持“缺陷查询”，它允许测试人员指定一组查询条件，如缺陷所属的功能、缺陷提交的日期、缺陷标题包含的关键词等，然后返回符合条件的缺陷报告。利用该功能，

测试人员可以查询自己负责的功能和相关功能的缺陷报告，浏览缺陷报告列表，阅读他感兴趣的报告（8.2.1节还介绍了一些定量分析缺陷报告的方法）。每天或每周阅读缺陷报告可以帮助测试人员更好地测试。

- **了解当前有哪些活跃的缺陷**。通过阅读缺陷报告，测试人员可以了解他的功能或他依赖的功能有哪些问题。这样，他发现相同的问题时不会重复提交缺陷报告。更重要的是，他知晓测试对象的局限，在测试时可以绕开一些阻碍，从而提高测试效率。
- **学习测试技巧，完善测试设计**。阅读缺陷报告，测试人员可以了解测试伙伴、内部用户、Beta版用户发现了哪些缺陷。如果他们发现的缺陷位于测试人员负责的功能，测试人员可以自问：为什么他发现了这个缺陷？他的测试设计有什么特点？为什么我的测试没有发现这个缺陷？如何增强我的测试策略，使我以后也可以发现此类问题？仔细地思考这些问题，记录思考结果，再应用于今后的测试，能够稳步提高测试质量。
- **获得测试灵感**。软件的功能具有相似性，在某个功能发现的缺陷，也许会在其他功能以另一种形式存在。测试人员参考相关功能发现的缺陷，能够产生新的测试灵感。例如，测试人员负责排版软件的图片测试。他知道图片和艺术字拥有一组相似的“特效”：阴影、倒影、柔化边缘、3D效果等，它们共享部分代码。于是，他阅读艺术字特效的缺陷报告，将某些测试设计“移植”到图片功能，从而使图片测试更具多样性。
- **追踪软件设计**。需求分析和功能设计都不可能完美，测试人员会在实际使用软件的过程中发现设计的缺陷。为了修正该缺陷，产品经理或程序员会修改功能设计。当新的设计缺陷浮现时，他们可能再次修改设计。随着项目进展，最初的规格说明将不能反映当前的设计，而新设计的讨论和决策都保存在缺陷报告中。面对这种情况，测试人员无需要求产品经理更新规格说明，因为将信息从缺陷报告搬运到规格说明费时费力，还可能丢失部分信息和可追踪性。最简单的方法是在缺陷标题中增加“[设计变更]”来标记此类缺陷（有些缺陷管理系统支持“关键字”字段，也可以将“设计变更”添加到该字段）。今后，他可以用关键词“设计变更”查询缺陷管理系统，以了解设计变迁的历史。
- **总结缺陷模式**。经过一段时间的积累，测试人员可以综合多个相似或相关的缺陷，提炼出它们的模式。针对这些模式，测试人员能够提出一些有针对性的测试策略，从而更快地发现该类缺陷。此外，一些缺陷模式暗示开发过程存在问题。例如，测试人员发现多个内存泄漏的缺陷报告，这可能暗示程序员之间的协作存在问题：当一个程序员调用其他同事的模块时，他并不了解该模块对于动态内存的编程约定，在进度压力下他凭经验完成了自己的代码，所以没有正确地释放该模块返回的内存。测试人员可以与开发经理当面讨论这些缺陷报告，帮助编程团队获得过程改进的信息。好的开发经理会感谢测试人员的总结，并采取相应的改进措施。

持续地回顾缺陷报告能让测试人员更好地理解产品、技术和团队。测试人员可以将一些研究成果用邮件、小组讨论等方式分享给测试小组。在潜移默化的影响下，阅读缺陷报告将成为一项团队的学习活动。

2.7　小结

本章以缺陷报告为中心介绍了一批测试实践，它们与测试人员的日常工作紧密相关。

- 提交缺陷报告的目的是让正确的缺陷得到修复。达成这一目标需要让缺陷评审者相信修复缺陷是正确的决定，并让缺陷修复者掌握足够的技术信息。
- 为了说服缺陷评审者，测试人员要通过测试去调查缺陷的普遍性（风险暴露的可能）和严重性（风险暴露的危害）。缺陷报告要提供足够的证据以说明该缺陷值得处理。
- 为了帮助缺陷修复者，测试人员要通过测试去获得更多的技术信息。对于难以重现的缺陷，测试人员要乐观地面对、缜密地测试、严谨地报告。
- 测试调查可能需要大量的时间，测试人员可以用时间盒来控制时间。
- 好的缺陷报告是独立的、准确的、便于查询的、完整的、易读的、客观的。
- 缺陷上诉的本质是说服别人采纳自己的建议，其核心工作是发掘和收集有说服力的证据，并通过交流将证据传递给缺陷评审者。
- 测试人员的“人脉”来自高质量的缺陷报告、高水准的信息服务和乐于助人的工作态度。测试人员要有意识地拓展“人脉”，让它为测试工作服务。
- 测试缺陷修复不要重复已有的测试，应该引入新的测试，在广度和深度上拓展测试设计。
- 定期阅读缺陷报告是一项有益的学习活动，能够帮助测试人员更好地测试。

第3章

测试文档

从测试工作的角度，测试文档大致可以分成两类：产品和工具[Kaner01]。

产品是供他人阅读和使用的文档。一些软件项目的合同要求项目团队在交付软件的同时交付一批规定的文档，其中的一些文档需要测试人员撰写。此外，某些软件过程认证对项目文档有明确的要求。为了通过认证，测试人员也要撰写规定格式的文档。

工具是测试小组的内部文档或测试人员的个人文档，其目的是帮助测试人员更好地测试。典型的例子是测试团队的内部Wiki和测试人员的笔记，它们内容宽泛、格式灵活、形式多样，专注于实际的测试工作，而不强求文档的完备与整齐。

对于大部分软件项目，项目团队的目标是交付有竞争力的产品，测试小组的任务是通过技术调查提供产品的质量信息。因此，测试人员不需要编写严格意义上的“产品”，即交付给客户和认证评估小组的文档。他们编写的文档都是团队或个人实施测试的工具，其根本价值在于帮助测试人员更有效地测试。偏离此价值的文档过程将浪费测试资源，且阻碍测试人员有效地工作。本章将围绕作为“工具”的测试文档，介绍一些编写和维护测试文档的原则与方法。

3.1 测试文档是持续演化的工具

作为工具的测试文档需要随项目进展而持续演化。本节概述测试文档的表现形式，并讨论编写和维护文档的基本方法。

3.1.1 测试文档是提供测试信息的一组文档

有些测试文档不但是测试小组的工具，也是项目团队的开发辅助工具。通过它们，测试小组将信息传递给项目团队和关系人，让他们可以更好地作决定。在编写文档时，测试人员不但要考虑到文档的实用性，还要考虑读者，为他们提供有价值的信息。常见的项目范围的文档包括以下几个部分。

- **测试计划**：记录了指导测试过程的一组想法[Kaner01]，概括地论述了产品特性、产品风险、测试任务、测试资源、测试进度、测试团队、团队协作等内容。在大型项目中，测试计划通常由测试经理撰写；在小型项目中，测试人员可以撰写一份文档，包含测试计划和测试设计的内容。测试小组、开发小组和产品经理会一起评审测试计划。通过评审，他们可以就测试小组的职责、开发小组的职责、测试与开发小组如何协作等关键问题达成一致意见。从这个角度，测试计划的撰写和评审是一项协作活动，测试计划是协作的成果。
- **测试设计规约**：记录了测试策略（一组指导测试设计的想法[Kaner01]）和详细测试设计。测试人员、程序员和产品经理会评审测试设计规约，查漏补缺，提出改进意见。在评审过程中，测试人员可以获得更多的测试想法，更好地理解软件；程序员可以获得测试信息，更好地编写开发者测试；产品经理能够了解测试覆盖，评估项目风险。
- **测试总结报告**：汇报执行了哪些测试、覆盖了哪些功能、获得了哪些信息、以及未来的测试行动等内容。通过阅读测试总结报告，测试经理、开发经理、产品经理等能够了解测试进度、产品问题、项目风险等信息。这为下一步的项目规划提供了基础。
- **缺陷报告**：也称测试事故报告[①]，记录了测试中发现的软件失败。通过缺陷报告，测试人员传递了产品的质量信息，并帮助开发小组修复软件缺陷。第2章已经专题讨论缺陷报告，故本章将不再讨论该类文档。

除了项目范围的文档，测试人员会编写一些测试小组或个人使用的测试文档，以帮助测试小组或自己完成测试任务。此类文档通常包括以下几种类型。

- **操作文档**：用详细步骤、屏幕截图、操作视频等形式记录完成某项任务的过程。测试涉及安装软件、配置测试环境、操作产品等任务。有时，这些任务可能很复杂，或需要特定的知识。操作文档使测试小组的所有成员都可以按图索骥，顺利完成任务。
- **测试笔记**：记录测试过程，包括测试设计、软件缺陷、测试发现、新测试想法等内容。在我的工作中，测试笔记是最重要的测试文档。我使用它记录测试想法，回顾测试过程，激发测试灵感。
- **测试资料**：是测试人员收集整理的测试支持材料。例如，测试人员需要测试一款Windows 8应用，于是他从微软网站获得了一份测试指南[Sundaram12]。他学习该测试资料，掌握测试Windows 8应用的基本策略，并根据当前应用的特点，精简出一份有针对性的测试文档。为了适应快速发展的软件行业，测试人员需要持续收集资料，通过积极阅读，将它们整理为面向实际工作的文档。
- **移交文档**：概述测试对象的特征，记录了前任测试人员的经验与建议。通过阅读该文档，继任的测试人员能够了解测试对象，获得相关测试资料，从而快速上手。
- **测试知识库**：记录测试小组的知识。它可以是Wiki网站，也可以是共享的笔记文件。

① 术语test incident report来自于IEEE 829国际标准（http://en.wikipedia.org/wiki/IEEE_829）。

可见，“测试文档”并不是一份文档，而是一组文档。因为软件开发是一项复杂的创新过程，涉及形形色色的活动与关系人，所以单一的文档满足不了多样化的需求。测试小组要根据项目实际情况，决定要编写哪些文档，以及这些文档要满足何种要求。

3.1.2　在测试中演化测试文档

测试是迭代过程，作为工具的测试文档也需要“与时俱进”。以测试设计文档为例，其价值在于准确地描述软件的特性、当前的风险、针对特性和风险的测试策略等。如果测试设计文档不反映软件的当前情况，不针对项目过程中浮现的风险，不体现测试人员通过测试获得的知识，那么按照此文档执行测试将损害测试的效果。或者，测试人员意识到文档已经过时，便将其束之高阁，凭自己的经验进行测试。这使得投入在文档上的努力付诸东流，也导致测试人员可能错过一些已知的重要测试。

为了让测试文档持续演进，最自然的方法是利用测试的迭代本质，在每一个迭代中“按需更新”测试文档。如图3-1所示，测试人员将测试设计、测试执行、测试评估、测试学习作为相互支持的活动，在项目过程中迭代地执行。在每一项活动中，他都可以更新测试文档，让文档更新成为测试的一部分。如果能做到“随风潜入夜，润物细无声”——将文档维护融入测试过程，那么测试文档就能成为“活的”文档，其维护阻力也能显著降低。

图3-1　在测试迭代中演化测试文档

最常见的测试迭代是测试新的构建。在此过程中，测试人员将测试文档作为行动的参考，将测试作为文档更新的驱动力。

- **测试设计**：在获得新构建后，测试人员回顾现有的文档，通过一些启发式问题来思考测试任务。这份测试设计完整吗？上次测试发现了什么问题或风险？针对这些问题和风险，需要调整测试设计吗？根据上次的测试结果，哪些功能只需简单测试，哪些功能需要重点测试，哪些测试不必运行，哪些测试需要深化？通过思考这些问题，他可以优化测试设计，并更新测试文档。

- **测试执行**：在运行测试时，测试人员会发现一些新信息，他会将重要的信息记录下来。例如，他学习到一些产品的知识，对于测试设计很有帮助，他就会写入测试笔记。再例如，他在测试执行中产生了新的测试想法，他就会补充到测试设计文档中。还例如，他发现特定的功能存在显著的风险，就会记录该风险，以便未来更深入地探索。
- **测试评估**：测试人员不但会评估软件的行为，还会评估测试的效果。在测试过程中，测试人员会多次查看测试笔记，以检查测试覆盖情况，并发掘更多的测试想法。由于软件的复杂性，许多信息仅在测试执行时才会暴露出来。测试人员应该利用这些信息，去完善测试设计，整理出未来的测试重点，并记录在案。
- **测试学习**：测试人员会根据测试反馈，去学习更多的知识，并整理成面向测试的文档。测试设计、执行和评估可以看作学习软件和技术的过程，测试人员会通过测试笔记和测试设计文档来记录所学到的知识。除此之外，测试人员还可能花时间学习一些专项知识。例如，因为被测系统包含数据库，测试人员会特意学习一些数据库的知识。学习要有明确的产出，否则很容易流于形式。为此，测试人员会根据所学内容编写一些测试资料，例如一份数据库常见软件错误列表或一份数据库测试的想法列表。

可见，测试文档与测试迭代是相互促进的。测试文档为测试学习、设计、执行、评估提供信息，帮助测试人员有效地测试；在测试迭代中，测试人员获得更多的知识，从而增强测试文档。这样的良性循环会稳步提高测试质量。

3.1.3　注重实效的测试文档

在阅读上节时，读者可能会产生疑惑：在测试过程中时刻维护测试文档会不会占用大量的测试时间，进而影响到测试效率？我的建议是：测试文档应该内容精炼、格式灵活、相互参考，从而尽可能降低维护的代价。

虽然测试文档具有多种形式，其核心仍旧是测试。对于测试人员而言，其内容不外乎：需要测试的功能和属性（测试对象和测试覆盖）、如何实施测试（测试设计与测试执行）、如何判断软件是否出错（测试先知）、测试过程中发现了哪些信息（测试发现）、测试进度和测试资源。其中，测试覆盖与测试设计（这两个概念在实际测试中紧密相关）是核心元素。在测试实战中，测试人员可以用测试覆盖大纲或测试想法列表为核心文档，用它来驱动测试设计和执行，并记录测试所发现的信息。

一种很简单的测试覆盖大纲是功能列表 [Kaner11]。如图3-2所示，PowerPoint允许用户在幻灯片中插入图片并编辑。对于PowerPoint的图片功能，测试人员可以建立如图3-3所示的PowerPoint图片功能符号列表对图片功能建模。限于篇幅，该功能列表只列出了PowerPoint图片的部分功能（深入的测试需要更详细的功能列表），但是它很好地体现了功能列表的特点。这些特点也是注重实效的测试文档所需要考虑的要素。



图3-2　PowerPoint的图片功能

- 插入图片（输入）
  - 图片格式
  - 图片尺寸
- 操作（"图片工具"下的命令）
  - 调整
  - ! 图片样式
  - 排列
  - 大小
- 应用（将图片与其他元素组合使用）
  - 文本
  - 形状
  - SmartArt
  - 图表
- 另存为图片（输出）
  - 被输出对象
    - 文本
    - 形状
    - SmartArt
    - 图表
  - 输出图片格式
- 打印含图片的文档
  - 物理打印机
  - ☆ 虚拟打印机
    - PDF
    - XPS
- 读写含图片的文档
  - 保存
  - 读取
  - ? 向后兼容

图3-3　图片的功能列表

第一，它建立了被测对象的整体模型。功能列表的顶层条目将图片功能划分为6个功能区域：输入、操作、应用、输出、打印和文件读写，这可以帮助测试人员全面地思考被测对象。测试人员可以分别测试每个功能区域，也可以构思测试情景去测试多个功能区域的交互。更重要的是，功能列表并不是一成不变的，随着测试的进展，测试人员可以根据新的认识调整功能列表的顶层条目（功能区域）。这样的调整体现了测试反馈对测试模型的修正，也说明测试文档既提供了测试灵感，又记录了测试所获得的新知。

第二，它提供了可扩展的测试设计框架。功能列表的层次结构让测试人员可以持续地补充细节。在某个条目下，他可以增加更详细的功能或针对该条目的测试想法。在初始测试设计获得基本的层次结构后，对功能列表的补充都是在测试迭代中"按需"发生的。当功能列表增长到一定程度后，测试人员可能发现原有结构并不理想，他会对其进行调整，例如增加新的顶层条目、将一些子条目移动到新的顶层条目等。调整结构说明测试人员的知识突破了原先的水平，新的模型更符合产品的本质。许多测试人员喜欢用思维导图来记录功能列表（见图3-4），它布局紧凑，能够一览全貌，从而更容易增加想法和调整结构。

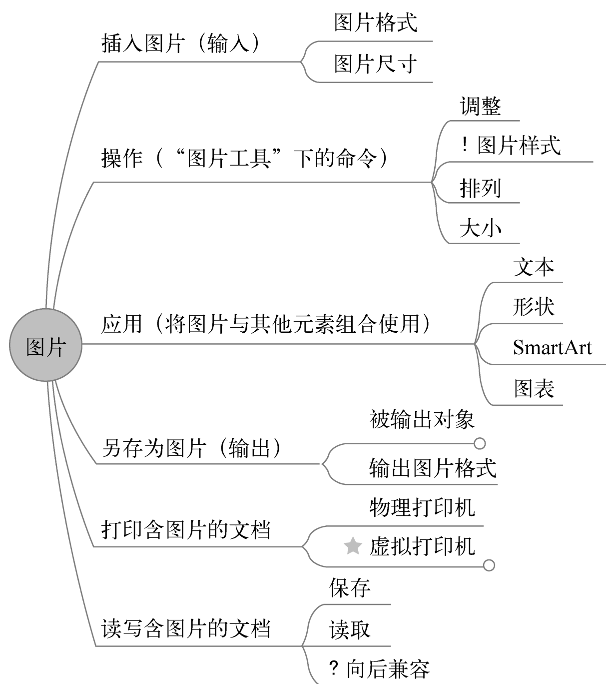


图3-4　功能列表的思维导图形式

第三，它提供了测试覆盖的目标。功能列表的主干是产品的功能，这提供了功能覆盖的目标。它也很容易纳入其他测试覆盖类型，例如"打印含图片的文档"包含子项目"物理打印机"和"虚

拟打印机”，这包含了平台覆盖的目标。

第四，它用简洁的形式提供丰富的信息。在图3-3中，“图片格式”和“PDF”都包含指向其他文档的链接。在测试这两个功能时，测试人员可以参考所链接的文档，获得更详细的信息。测试需要一组文档的支持，核心文档只记录必要内容，更详细的信息可以链接到其他文档。这样做可以保持核心文档的简洁，让测试人员快速地掌握全局，并降低最常用文档的维护开销。

文档相互链接是一项好的实践。这让多份文档相互支持、紧密联系，使之成为互为补充的文档集。测试人员对软件的认知总是逐步增强的，补充新的文档，让它与已有文档相互联系，是一个渐进的学习过程，其结果是一个逐渐完善的领域模型。

第五，它格式灵活，允许用多种方式记录信息。功能列表的条目通常是词语或语句片段，配合助记符（如图3-3中的问号），能让测试人员快速记录测试想法和测试发现。在测试迭代中，此类灵活且非正式的表达通常更高效。例如，测试人员可以定义自己的助记符，用自己最习惯的方式记录信息。这体现了测试文档的“工具”属性——它应该帮助测试人员，而不是限制甚至阻碍他的发挥。如果测试人员需要提交正式的测试报告，他可以根据词语和助记符，补充记录测试范围、测试设计、测试结果和总体评估。

功能列表并不是组织测试想法的唯一方法。不同的测试任务有不同的需求，需要不同形式的核心文档，以下是一些例子。

- 功能测试可以以功能列表为骨干、以测试想法为叶节点，其表现形式是符号列表或思维导图。
- 测试一个产品补丁时，测试人员可以从测试想法的思维导图中抽取若干分支，构成新的思维导图。测试补丁不需要完整的思维导图，不过常常要将几个分支合在一起考虑，以周密地测试产品的一个部分。
- 浏览器兼容性测试可以以“功能—浏览器”矩阵为核心，其表现形式通常是表格（详细案例参见3.2.5节）。
- 在安全测试时，测试人员可以从一份安全测试的缺陷目录开始测试（缺陷目录的讨论参见3.2.12节）。该缺陷目录记录了典型缺陷、失败症状、攻击手段、可用工具等信息。根据这些信息，测试人员可以构造测试用例，去挖掘隐藏的缺陷。

可见，测试任务的多样性导致测试文档的多样性。在一个复杂的项目中，测试人员很可能需要执行多项测试任务。面对此类情况，维护若干份目标明确的简洁文档，其效率要优于维护一份无所不包的长篇文档。测试人员可以参考如下的文档维护策略。

- **在项目早期，编写测试设计规约，概述测试策略**。邀请产品经理、程序员、测试经理等进行评审，并根据反馈意见做出修改。
- **在测试设计规约的指导下，为测试活动收集、复用、编写、改进测试文档**。在特定的测

试活动中，测试人员可以复用与修订测试设计规约，可以编写一页篇幅的简洁文档，也可以即兴测试。总之，从实战出发，只做必要的文案工作。

- **将文档集中管理，并周期性地备份**。可能的做法包括：将文档提交到源代码管理系统、将文档保存在共享目录、将文档保存在内部网站、将内容写入共享的测试知识库等。测试文档是重要的测试资产，值得慎重保管。
- **请同事评阅文档**。在生成一幅思维导图或一个配置矩阵后，测试人员可以用邮件评审或桌面走查的方式，邀请相关同事评阅测试设计。与评审十几页的文档相比，检查一页篇幅的文档开销很低，却能够快速提供有价值的意见。如果测试人员和同事面对面地讨论，通过头脑风暴在短时间内可以激发出更多的测试想法。
- **在项目的重要里程碑完成时，整理已有文档**。可能的做法包括：将一些有价值的内容移入测试设计规约、链接相关文档、补充测试知识库、编写测试移交文档等。

作为工具的测试文档，其价值取决于它能够帮助测试人员组织测试并找到缺陷的程度[Kaner99]。所谓“注重实效的文档”就是聚焦于文档的基本价值，舍弃不必要的修饰。一份直抵核心的文档更容易编写、阅读、修改和流传。

3.2　形形色色的测试文档

本节介绍一些常见的测试文档。它们可以单独成文，也可以组合在一起构成更周详的文档。虽然形式不同，但是功能主要有以下3个[Kaner99]。

- **为完成技术任务提供便利**：测试文档提供技术信息，以支持更好地测试设计和执行。
- **改善测试任务和测试过程之间的联系**：测试文档解释测试策略背后的思想、测试的覆盖范围、测试工作的规模、测试的深度和广度、测试分工等，从而帮助测试小组与整个项目团队交流和协作。
- **为组织、规划和管理测试项目提供结构**：测试文档提供管理结构，让测试经理更好地管理测试项目，让测试小组更好地为项目服务。

3.2.1　测试计划

测试计划是一组指导测试过程的想法[Kaner01]。制订良好的测试计划需要深入评估项目情况，充分利用测试资源，合理安排测试任务。在此过程中，测试人员可以参考James Bach提出的启发式测试计划的语境模型（Heuristic Test Planning Context Model，HTPCM）[Bach00a][Kaner01]。

图3-5展示了HTPCM的基本元素，其中测试过程是核心，测试任务决定了测试过程的目标。计划制订者要根据语境（开发、测试团队、测试实验室、需求）来决定测试过程和测试任务。

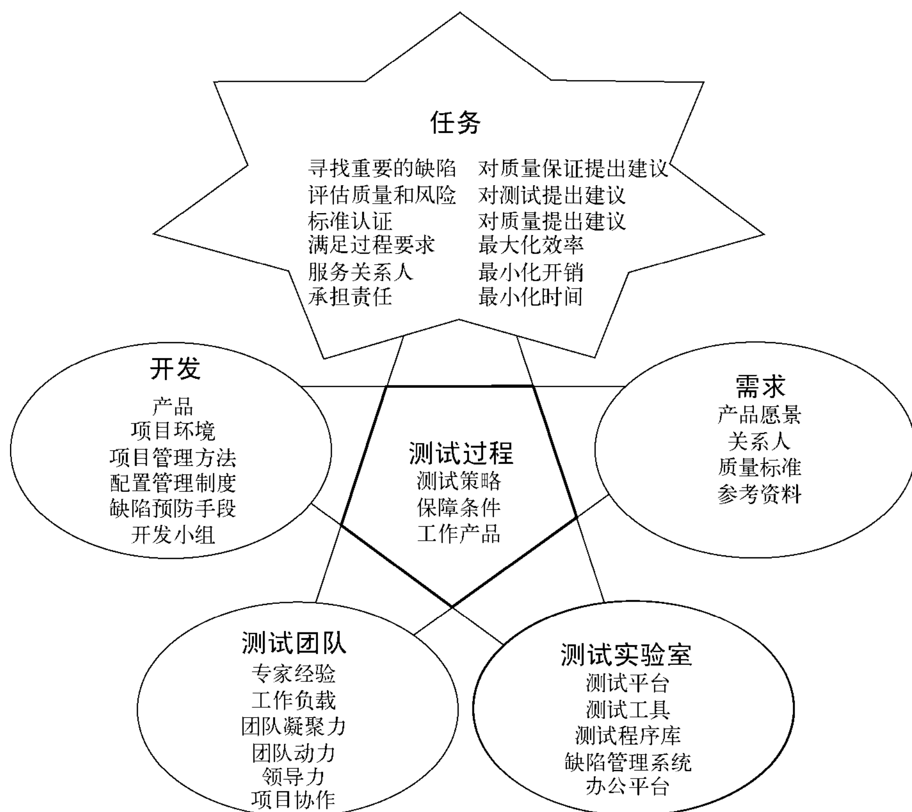


图3-5　启发式测试计划模型

- **需求**：是产品获得成功的条件，包括产品愿景、关系人的期望、质量标准、参考资料等。计划测试的过程也是一个学习需求的过程，只有理解需求才能正确地设定测试目标。
- **开发**：是产品开发的环境，包括被测产品、项目周期、项目管理方法、配置管理制度、缺陷预防手段、开发小组等。开发环境对测试小组的工作方式有很大影响。测试小组要推动开发环境的改变，让它更有利于测试，同时也要积极地适应它，利用积极因素，缓解不利因素。
- **测试团队**：是测试小组的特点，包括专家经验、工作负载、团队凝聚力、团队动力、领导力、项目协作等。
- **测试实验室**：是测试小组可使用的软硬件资源，包括测试平台、测试工具、测试程序库、缺陷管理系统、办公设备等。
- **任务**：是测试小组要提供的服务，可能的选择包括寻找重要的缺陷、评估质量和风险、标准认证、满足过程要求、服务关系人、承担责任、对质量保证提出建议、对测试提出建议、对质量提出建议、最大化效率、最小化开销、最小化时间等。测试资源总是有限的，测试小组要确定可承担的服务，并与整个项目团队交流，以达成一致意见。只有提

供正确的服务，测试小组的付出才是有价值的。

- **测试过程**：是完成测试任务所需要执行的活动，其要点是测试策略、保障条件、工作产品。
  - **策略**：是实施测试的指导思想。测试计划需要讨论测试内容、测试重点、测试优先级、测试方法等内容。
  - **保障**：阐述如何利用测试资源实现测试策略，包括谁做测试、何时测试、需要何种软件、需要何种硬件等。
  - **产品**：是测试过程的产出。常见的产品包括测试设计规约、自动化测试用例、测试工具、缺陷报告、测试总结报告。测试小组需要与服务对象讨论，明确产品的类型和要求。

在HTPCM的指引下，测试计划者根据语境来完成测试计划[Bach00a][Bach99]。

- 了解谁是测试小组的服务对象、他们重视什么、需要什么服务。
- 了解并协商当前项目对测试工作的要求和约束，以及能获得的测试资源（人员、时间、软件、硬件等）。
- 分析产品和产品风险，建立产品和风险的初步模型。
- 协商并确定测试任务。
- 根据要求、约束、资源和任务，制定测试过程。其中，测试任务决定测试策略和测试产品，测试策略指导保障安排，而保障安排所能使用的资源又约束了测试策略的选择——测试策略与保障安排相互依赖且彼此影响。
- 分享并交流测试计划，使整个项目团队理解测试安排，并获得相关反馈。
- 监控项目情况，并持续调整测试过程，使项目、任务和测试过程可以和谐一致。

HTPCM的特征是测试计划者通盘考虑产品需求、项目环境、测试小组、测试资源，明确测试任务，并制订相应的测试过程。它认为好的测试计划是基于语境的，只有符合项目情况的测试方案才能有效地工作。正如软件专家Gerald M. Weinberg所说，测试方法各有巧妙，无所谓好坏。讨论“优秀”，不应该只考虑测试方法本身，而是要考虑测试、实现和环境是否紧密联系，即“优秀”是对测试、实现和环境之间关系的描述[Weinberg08]。

作为测试计划的成果，测试计划文档记录了测试小组的选择与安排。表3-1引用了Chrome OS测试计划的章节标题 [Whittaker12]。不难看出，测试计划的作者Jason Arbon将测试过程与开发过程紧密结合，并根据语境作出了一系列决策。

表3-1　Chrome OS测试计划

章节标题	在HTPCM中的概念	所讨论的内容
总论	任务、测试过程之策略	总论测试小组的任务和基本策略 ● 使用基于风险的测试策略 ● 在目标硬件矩阵上运行测试自动化

（续）

章节标题	在HTPCM中的概念	所讨论的内容
总论	任务、测试过程之策略	● 支持快速的开发迭代 ● 开放工具和测试的源代码 ● 将Chrome OS作为Chrome浏览器的首要平台（优先级高于其他操作系统） ● 测试的任务是提供质量相关数据 ● 关注可测试性
风险分析	开发之产品	测试小组将负责功能性风险分析，包括风险分析的目的、方法和产出
针对所有构建的基线测试	测试过程之策略	每个构建都要执行的测试：冒烟测试和性能测试
针对最新可测构建的每日测试	测试过程之策略	每天对于最新的可测构建进行持续测试 ● 手工的验收测试 ● 自动化功能性回归测试 ● 运行最热门 web app的测试用例 ● 压力测试、持续测试、稳定性测试 ● 持续的手工探索式测试和漫游测试 ● 针对自动更新的测试
针对发布候选构建的测试	测试过程之策略	对于发布候选构建的测试 ● 网站兼容性测试 ● 演示情景验证 ● P0缺陷验证 ● 全面压力和稳定性测试 ● 手工测试
平衡手工测试和自动化测试	测试过程之策略	解释手工测试和自动化测试的适用范围，为测试资源运用提供指导
开发与测试小组的质量责任	开发之开发团队	解释开发小组和测试小组各自承担何种质量责任与活动
发布通道	开发之项目周期、开发之项目管理	概述产品发布的阶段与过程
用户反馈	需求之关系人	如何获取用户反馈和组织他们的数据
测试用例仓库	测试过程之产品	如何保存手工测试用例和自动化测试用例
测试仪表盘	测试过程之产品	测试小组将质量数据用可视化的方式展示出来
虚拟化	测试实验室之测试工具	测试小组将利用虚拟化技术来提高测试效率
性能	测试过程之策略	概述性能测试策略
压力、持续、稳定性	测试过程之策略	概述压力测试、持续测试、稳定性测试的测试策略
测试执行框架	测试实验室之测试程序库	解释了开发和测试小组所使用的测试执行框架Autotest
设备厂商	需求之关系人、 测试过程之保障	解释了测试小组如何与OEM厂商合作
硬件实验室	测试实验室之测试平台、 测试过程之保障条件	概述测试所需要的硬件资源

（续）

章节标题	在HTPCM中的概念	所讨论的内容
端到端测试大规模自动化	测试过程之策略	解释了端到端测试的测试策略，其目标是覆盖大量的软硬件组合
测试浏览器的应用管理	测试过程之策略	解释如何测试浏览器的应用管理功能
浏览器可测试性	开发之产品	阐述如何实现浏览器可测试性 ● 将Selenium和WebDriver移植到Chrome OS ● 通过JavaScript DOM来暴露Chrome的用户界面和功能 ● 提供更高抽象水平的自动化脚本
硬件	测试过程之策略	解释测试小组所测试的硬件功能
时间线	开发之项目周期	解释了5个里程碑（2009年4季度、2010年1季度、2010年2季度、2010年3季度、2010年4季度）分别要达成哪些目标
主要测试驱动角色	开发之项目管理	列举了测试活动的驱动角色，他们会领导特定的测试活动
相关文档	测试过程之产品、测试过程之保障	列举了相关测试支持文档

值得一提的是，Chrome OS测试计划只有10页（B5幅面）。对于如此规模的产品，这确实是一份非常精炼的文档。这样做是为了让每位测试人员都能读完测试计划，并立即了解最重要的策略和安排。对于一个大型项目，如此短小的纲领性文件有助于形成团队共识，并聚焦重点。至于更详细的技术内容，该文档提供了一组外部链接（相关文档），供测试人员有选择地参考。这些相关文档通常由测试驱动者编写，能够为实际测试提供有力支持。

3.2.2 Google ACC

ACC（Attributes, Components, Compatibilities）是Google测试团队使用的一种建模方法，用来快速地建立产品的模型，以指导下一步的测试设计 [Whittaker12]。Google的工程师还开发了支持ACC的Web应用，并将其开源[①]。

运用ACC建模的第一步是确定产品的属性。属性是产品的形容词，是与竞争对手相区别的关键特征。按照敏捷开发的观点，属性是产品交付的核心价值。

《Google软件测试之道》（*How Google Tests Software*，简称HGTS）详细介绍了Google+的ACC模型，其中Google+的属性如下。

- **社交：**鼓励用户去分享信息和他们的状态。
- **表现力：**用户可以运用各种功能去表达自我。

① 详见http://code.google.com/p/test-analytics/。

- **自如**：让用户以直观的方式做他们想做的事。
- **相关**：只显示用户感兴趣的内容。
- **可扩展**：能够与Google的已有功能、第三方网站和应用集成。
- **隐私**：用户数据不会泄漏。

ACC以属性开始，是产品竞争的自然选择，也符合Google的开发实践。在Google的项目中，程序员和测试人员的比例通常是10∶1或更高。程序员会编写大量的自动化测试用例，对产品实施周密的测试，因此测试人员主要关注用户价值和系统测试。即便如此，测试人员也没有足够的资源测试所有用户行为。因此，测试人员通过属性来明确产品的核心价值，从而区分出测试对象的轻重缓急。获取属性的信息源可以是产品经理、市场营销人员、技术布道者、商业宣传材料、产品广告等。测试人员也可以使用"卖点漫游"来发掘和检验产品的卖点[Whittaker09]（5.4节将讨论漫游测试）。

第二步是确定产品的部件。部件是产品的名词，可以理解为产品的主要模块、组件、子系统。HGTS给出的Google+部件如下。

- **个人资料**：用户的账户信息和兴趣爱好。
- **人脉**：用户已经连接的好友。
- **信息流**：由帖子、评论、通知、照片等组成的有序的信息流。
- **圈子**：将好友分组，例如把不同的好友归于"朋友"、"同事"等小组。
- **通知**：当用户在帖子中被提到时，向他显示提示信息。
- **视频群聊**：视频对话的小组。
- **帖子**：用户和好友发表的信息。
- **评论**：对帖子、照片、视频等的评论。
- **照片**：用户和好友上传的照片。

部件可以看作功能列表的顶层元素，是产品核心功能的清单。《Google软件测试之道》建议部件列表要尽可能简单，10个部件很好，20个就太多了。其目的是重点考虑对产品、对用户最重要的功能与代码，并避免漫长的部件列表所导致的分析瘫痪。

第三步是确定产品的能力。能力是产品的动词，描述了部件如何实现属性。HGTS给出的Google+能力矩阵如表3-2所示。

表3-2　Google+的能力矩阵

	社　交	表　现　力	自　如	相　关	可　扩　展	隐　私
个人资料	在好友中分享个人资料和兴趣爱好	用户可以在网上表达自我	很容易创建、更新、传播信息		向被批准的、拥有恰当访问权限的应用提供数据	用户可以保密隐私信息。只向被批准、拥有恰当访问权限的应用提供信息

（续）

	社　交	表　现　力	自　如	相　关	可　扩　展	隐　私
人脉	用户能够连接他的朋友	用户可以定制个人资料，使自己与众不同	提供工具让管理好友变得轻松	用户可以用相关性规则过滤好友	向应用提供好友数据	只向被批准、拥有恰当访问权限的应用提供信息
信息流	向用户提示其好友的更新			用户可以根据兴趣过滤好友更新	向应用提供信息流	
圈子	将好友分组	根据用户的语境创建新圈子	鼓励创建和修改圈子		向应用提供圈子数据	
通知			简明地展示通知		向应用提供通知数据	
视频群聊	用户可以邀请他们的圈子加入群聊。用户可以公开其群聊。好友访问用户的信息流时，他们被告知群聊	加入群聊前，用户可以预览自己的形象	只要几次点击就可以创建并加入群聊。只要一次点击就可以关闭视频和音频输入。可将好友加入已有的群聊		用户可以在群聊中使用文字交流。YouTube 视频可以加入群聊。在“设置”中可以配置群聊的硬件。没有摄像头的用户可以音频交谈	只有被邀请的用户才能加入群聊。只有被邀请的用户才能收到群聊通知
帖子		表达用户的想法			向应用提供帖子数据	帖子只向被批准的用户公布
评论		用评论表达用户的想法			向应用提供评论数据	评论只向被批准的用户公布
照片	用户可以分享照片		用户能方便地上传照片。用户能方便地从其他来源导入照片		与其他照片服务集成	照片只向被批准的用户公布

能力通常是面向用户的，反映了用户视角的产品行为。测试人员也要保持能力矩阵的简洁，他们应该关注对用户而言最有价值、最有吸引力的能力，并在合适的抽象层次阐述能力。最重要的是，能力应该是可测的，测试人员能够设计测试来检查产品实现了预期的能力。

有了能力矩阵，测试团队就完成初始的测试计划。前Google测试总监James Whittaker称之为“10分钟测试计划”[Whittaker11]。其基本思路是专注于核心属性、核心功能和核心能力，省略一切不必要的细节。之后，测试团队会利用矩阵去指导测试设计，通常矩阵中的一条能力对应一个测试对象、测试策略或测试情景，而复杂的能力会演化出更多的测试设计。

Google所提供的开源Web应用可以分析项目信息，包括测试用例、代码变更、产品缺陷等，以确定能力矩阵中的高风险区域。图3-6引用自James Whittaker在GTAC 2010①的闭幕演讲的幻灯片[Whittaker10]，是Chrome OS的能力矩阵的热点图。图中绿色表示低风险区域，红色表示高风

① 5th Annual Google Test Automation Conference, Hyderabad 2010, http://sites.gtac.biz/gtac2010/。

险区域，粉红色和橙色则表示风险居于前两者之间。测试人员可以根据热点图，更好地确定测试优先级，将有限的资源运用在最需要的地方。

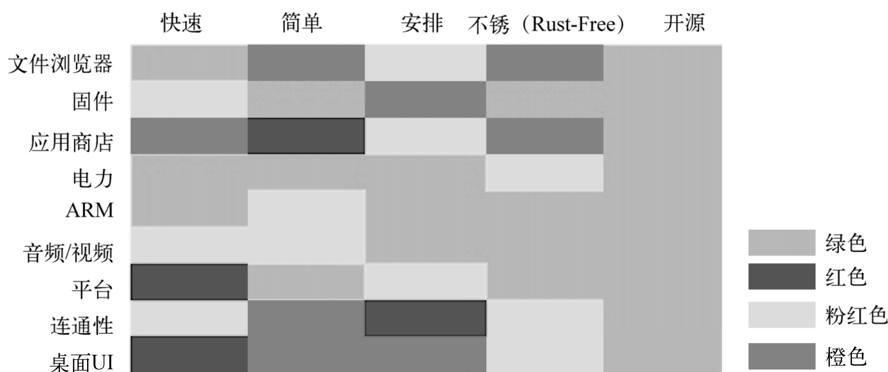


图3-6　Chrome OS的热点图

许多团队的风险分析依赖于测试人员的经验和猜测，Google的ACC工具则通过分析项目元素（测试用例、代码变更、产品缺陷等）来识别风险。即便不使用Google的工具，测试人员也可以利用电子表格记录能力矩阵，并自行计算各个条目的风险（一些Google的测试人员也是这么做的）。在评估风险时，他可以考虑以下因素（更多讨论参见第8章）。

- **自动化测试用例**：该区域有自动化测试用例吗？测试在定期运行吗？测试通过率是多少？测试用例覆盖了哪些方面，没有覆盖哪些方面？
- **手动测试**：有人手动测试该区域吗？经过测试，他们对该区域有信心吗？如果满分是10分，他们会打几分？
- **代码变更**：该区域近期存在代码变更吗？变更频繁吗？变更是新增功能、代码重构、还是缺陷修复？
- **代码复杂度**：代码的规模是多少？代码是否复杂？如果复杂度的满分是10分，该区域的代码能得几分？
- **产品缺陷**：该区域的缺陷多吗？有哪些典型缺陷？哪些缺陷已经被修复？哪些缺陷还没有被修复？活跃的缺陷是在快速增加还是稳步下降？

在计算此类风险因素时，测试人员可以采用尽可能简单的度量方法。一方面，简单的方法更容易解释度量值的含义，有助于针对度量值采取相应的行动。另一方面，复杂的方法增大了分析的难度，却往往不能提供更多的收益。通过测试去获得直接的反馈，并定期重新评估风险，是更注重实效的方法。这也符合ACC的风格：快速地前进、持续地迭代。在测试计划时，测试人员只要快速地确定能力矩阵，而不必担心遗漏。随着测试的进展，他会对矩阵作出必要的调整，以优化测试的价值。

3.2.3　测试设计规约

测试设计规约记录了测试人员的测试设计。其内容和形式往往由测试小组决定，许多测试小组还制定了相应的文档模板。本节介绍一些测试设计规约的编写技巧，供测试人员参考。

第一，测试设计规约是一个“容器”，可以容纳各种测试模型和资料。本章所介绍的测试文档，如功能列表、思维导图、Google ACC等，都可以纳入同一份测试设计规约，以便从不同角度、不同层面来描述测试设计。

测试人员应该用最直观、最方便的形式表达测试设计。对于某些测试设计，符号列表（和思维导图）的层次结构很方便；对于另一些设计，表格（如Google ACC的能力矩阵）所提供的二维结构就很直观；在某些情况下，UML等图形化的表达更有表现力。可见，并不存在普遍适用的测试记录方式，测试人员需要用不同的方式记录不同的测试设计。作为测试设计的主要载体，测试设计规约需要容纳多样化的测试表达，也可以用链接指向相关测试文档。

第二，测试人员需要建立测试设计的框架。首先，他要列举测试类型，例如功能测试、安全测试、性能测试、易用性测试等。然后，针对测试类型，列出主要测试想法，例如图3-3的功能列表（和图3-4的思维导图）列举了图片功能测试的主干和分支。

清晰的框架让测试人员和评审者在整体上把握测试设计，知晓测试所覆盖的范围。如果设计遗漏了某种测试类型，评审者能够快速发现该问题，并提出改进意见。此外，在测试学习、设计、执行、评估的迭代中，测试人员会逐渐掌握更多的软件和测试知识。开放性的框架提供了可扩展的结构，方便测试人员补充测试想法，增强整体测试设计。

第三，测试想法优于测试用例。测试用例详细地记录了测试设计，包括测试环境、执行步骤、检查方法、预期结果等。在软件项目中，测试人员会执行许多测试用例，有些测试用例往往只运行一次，详细地记录所有测试用例将占用过多的项目时间，压缩了实际测试的时间。而且，随着项目的发展，软件设计常常会发生变动，这导致许多测试用例不再有效。详细地记录测试用例势必导致沉重的文档维护任务。在进度压力下，一些测试人员选择不再维护文档，而逐渐过时的文档最终会被遗弃。

为了降低编写和维护测试设计的代价，我建议测试人员优先记录测试想法，只在必要时添加详细描述。例如，必应词典客户端从服务端获得单词的中文解释，服务端所返回的是一份XML文档。针对服务端可能返回错误的XML文档，测试人员可以记录如下测试想法。

- 服务器宕机，没有返回任何消息。
- 服务端返回的字符串不是XML文档。
- 服务端返回的字符串是XML文档，但是该XML文档不符合必应词典所定义的XML Schema。

- 服务端返回的XML文档符合预定义的XML Schema，但是该文档包含错误。重点考虑的错误：单词的中文解释为空字符串、单词的中文解释包含非法字符、单词的中文解释超长。
- 如果有恶意网站伪装成必应词典服务端向客户短发送恶意XML文档，客户端能否妥善处理？

这些测试想法没有记录测试环境、执行步骤、检查方法和预期结果等如何测试的信息，只简要地描述了测试动机（检查必应词典可以处理服务端的错误）和测试内容。这让测试人员用较短的时间记录更多的测试设计，也简化了测试评审的工作。在测试迭代中，测试人员可以补充新的测试想法，也可以加入一些如何测试的细节，让测试过程产生有价值的文档。

第四，合理地使用文档模板。许多团队都提供了测试设计规约的文档模板，许多测试人员也希望获得模板来简化文档工作。不过，模板只是产生文档的工具，它不能取代思考和交流。如果根据模板亦步亦趋地填写其中的空白，测试人员可能被它束缚了思路，所产生的文档“看上去很好”，但是缺乏多样性的测试想法。

为了更好地设计测试，一些测试人员在设计之初并不使用模板，而是将测试想法写在白纸或空白文档中[Kelly11a][Kelly11b]。他们尽可能记录各种测试想法，而不考虑记录格式或其他约束。当大部分测试设计完成后，他们将测试设计移入模板，以获得符合项目要求的文档。在此过程中，他们用模板去检查测试设计的完整性，并补充之前遗漏的测试想法。

3.2.4　功能列表

3.1.3节以PowerPoint的图片功能为案例，介绍了功能列表。这是一种功能测试的建模方法，通过将产品的功能分解为层次结构，为功能覆盖提供指导。

功能列表与漫游测试可以相互支持。漫游测试是一组以漫游隐喻为核心的测试方法[Kaner11]（5.4节将详细讨论漫游测试）。在应用漫游测试时，测试人员常常会游历被测产品的功能、结构和数据，并运用他的技能和经验去发掘产品的缺陷。

一方面，功能列表为功能漫游提供了有益的信息。它像一幅地图，既描绘了产品的概况，又提供了必要的细节，为探索者提供了指南与参考。测试人员可以漫游功能列表上的所有元素，以实施全面探索；也可以选择遍历某个功能子集，以实施专项测试。

在另一方面，倘若测试人员不了解被测产品，他可以通过功能漫游来建立功能列表。从这个角度，漫游的过程就是测试建模的过程，功能列表就是测试建模的产出。测试专家Cem Kaner建议，在软件尚不成熟时，测试人员应该同情地测试[Kaner01] 。此时，测试的目的不是发现所有缺陷，而是建立测试模型，发现风险区域，为今后的测试奠定基础。对此，测试专家Michael Bolton有一番精彩的论述：“同情地测试非常重要，一些测试人员会忍不住全力寻找缺陷。但是Jon Bach[①]

① Jon Bach和James Bach是兄弟，他们都是知名的测试专家。

说在探索式测试的早期，要通过测试来发现产品的优点。我觉得这很奇怪，他指出，如果只是寻找并记录缺陷，就不能专注地漫游产品并构建产品的模型，我才恍然大悟。”[Kaner11]

测试人员需要建立产品的大局观，同时掌握产品的优点、缺点、概念模型和实现逻辑。漫游测试是很好的学习过程，功能列表是一个有益的学习成果。

在测试设计与执行时，测试人员可以将功能列表视作覆盖率指南。他逐个检查每个功能，阅读相关的测试想法，从而产生新的测试想法。在此过程，他可以自问：

- 该功能与当前测试任务相关吗？
- 该功能存在什么风险？可能会有什么缺陷？
- 通过什么测试可以发现这些缺陷？
- 在上次测试中，该功能表现如何？已有的测试想法，哪些值得再次尝试？哪些不必再测？
- 依据当前的进度和资源，如何实施这些测试？
- 功能列表是否充分？有没有漏掉一些功能？

另一种更有效的方法是综合功能列表中的多个元素，来测试功能的协作。随着产品逐渐成熟，隐蔽的缺陷往往存在于功能的组合，暴露于复杂的流程。这要求测试人员综合多方面的信息，来更深入、更多样地测试系统。当测试人员考虑功能的组合时，他可以自问：

- 该功能与哪些功能相关？
- 功能的组合有没有揭示出新的风险？可能会有什么缺陷？
- 哪些功能访问同一批数据？哪些是生产者？哪些是消费者？
- 如何设计测试来同时测试这些功能？
- 如何构造一个有意义的业务流程，让它能够访问尽可能多的功能与数据？
- 对于相互依赖的功能，某个功能的失败是否对其他功能造成恶劣影响？

在回归测试时，功能列表是很好的参考。例如，测试人员可以按如下测试策略对PowerPoint的图片功能（如图3-3所示）进行回归测试。

(1) 新建一个PowerPoint文档。

(2) 向文档中插入图片。

  - 覆盖所有支持的图片格式。
  - 覆盖典型的图片尺寸。
  - 覆盖来自单反相机的大型图片 。

(3) 操作文档中的图片。

  - 覆盖“图片工具”下的所有命令。

- 用一个命令修改图片。
- 用多个命令修改图片。
- 保持一些图片不被修改。

(4) 应用文档中的图片。

- 将图片与其他元素组合使用。
- 覆盖文本框、形状、SmartArt、图表、日期与时间等元素。

(5) 将文档中的元素导出为图片。

- 覆盖所有可以被输出的元素：页面、图片、形状、SmartArt、图表等。
- 覆盖所有支持的图片格式。
- 覆盖相关的用户界面命令：上下文菜单（即右键弹出菜单）、Backstage（点击“文件”→“导出”→“改变文件类型”）、复制粘贴到画图板。
- 覆盖相关的Visual Basic Application对象模型。

(6) 打印该文档。

- 打印到黑白打印机。
- 打印到彩色打印机。
- 打印到PDF文档。
- 打印到XPS文档。

(7) 保存该文档并重新打开。

- 另存为所有支持的格式。
- 用PowerPoint打开生成的文档。
- 用旧版本PowerPoint打开生成的文档。

利用该测试策略，测试人员可以用一个很长的流程覆盖大多数的图片功能，不但测试了图片与其他功能的组合，还测试了程序的稳定性和资源占用。测试结束时，测试人员可以获得一个大型的PowerPoint文档，它包含各种图片和元素，为今后的回归测试提供良好的测试数据。

3.2.5 大纲与思维导图

功能列表用层次结构表达软件的功能。与之类似，测试人员可以用层次结构组织测试想法，使得测试想法的分类关系一目了然。

我习惯用大纲来记录测试想法。在文档编辑器（如Microsoft Word）中，符号列表、编号列表和多级列表都可以方便地编写和修改测试想法的大纲。

此外，许多测试人员喜欢用思维导图来记录测试想法，也获得了很好的效果。图3-7是测试专家Darren McMillan制作的思维导图形式的测试计划[McMillan11]。限于篇幅，该图仅呈现了原测试计划的局部[①]，但仍旧可以看出思维导图的一些重要特点。

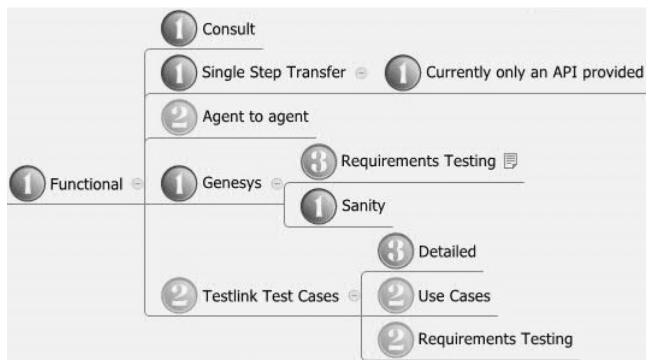


图3-7 功能测试的思维导图（局部）

- 与大纲一样，思维导图提供了清晰的层次结构。沿着图中线条，某个想法衍生出更具体的想法，而这批想法又继续衍生出更细致的想法。这种清晰的结构让测试人员方便地把握测试想法的分类与层次。
- 思维导图用词语来记录测试设计，这使测试人员专注于思考。由于无需写下不必要的细节，测试人员可以用较短的时间记录大量的测试想法。当测试人员快速思考、积极联想时，他能够获得多样化的测试设计。
- 图3-7的“需求测试”拥有一个注释，它包含更详细的测试设计。有时，词语或短句并不能很好地传递测试想法，必要的注释能够进一步解释测试思路，消除误解。在这一点上，电子的思维导图既保留了传统的思维导图的简洁，又拥有更丰富的信息。
- 与大纲相比，思维导图更“引人注目”。图3-7使用了形状、线条、颜色、维度、质地等视觉元素，它们激发了大脑的兴趣，这是单纯的文字列表难以做到的。当测试人员对一幅图感兴趣时，他会更积极地思考，从而产生更多的想法。这对于测试计划的编写、评审、修订都有促进作用。
- 思维导图可以使用颜色、标号等视觉元素来表达语义，这提高了信息传达的效率。如图3-7所示用红色的①、黄色的②、蓝色的③表示测试想法的优先级，使得测试优先级一目了然。
- 大纲和思维导图都鼓励测试人员扩展已有的结构，持续地加入想法。经过一段时间的努力，测试人员可以积累一大批相互关联的测试想法。此时，大纲的呈现会遇到一定的困难。对比图3-3的大纲和图3-4的思维导图可知，对于数量相同的测试想法，大纲的长度更

① 完整的思维导图：http://www.bettertesting.co.uk/content/?p=956。此外，ministryoftesting.com也提供了许多软件测试的思维导图。

长，这使得同一层次的测试想法在视觉上疏离，且没有充分利用屏幕的宽度。反观思维导图，其结构更紧凑，随着想法的加入，它向两侧延伸，更好地利用了屏幕的宽度。随着宽屏显示器的普及，思维导图在排版上的优势更明显。

除了测试计划和设计，测试人员还可以将思维导图应用于测试报告、信息组织、自我管理等领域[McMillan11]。作为一款思维辅助工具，其应用领域几乎是无限的。

3.2.6 表格（矩阵）

在测试设计中，表格适合表达两个变量之间的关系或两个变量共同作用的结果。例如，表3-2从软件能力的角度表达了属性与部件的关系。又例如，表3-3 是浏览器兼容性测试的配置矩阵，所记录的测试结果可以看作是浏览器X与功能Y共同作用的结果。

表3-3 浏览器兼容性测试的配置矩阵

	桌面IE 10	沉浸式IE 10	IE 9	IE 8	Firefox	Chrome
功能1	✓	✓	✓	✓	✓	✓
功能2	✓	Bug1，Bug2	✓	✓	Bug3	Bug4
功能3	✓	✓	✓	✓	✓	✓

在测试设计时，测试人员用表3-3规划即将实施的测试，估算完成全部测试需要的时间。在表格下方，他可以记录每个功能的测试想法，这些想法可以在多个浏览器中复用。在测试执行时，他可以在表中记录测试结果，例如所发现缺陷的ID。在测试完成后，他对表格稍作整理，将其作为测试总结报告的一部分发送给测试经理。测试经理可以很直观地了解测试情况，包括哪些功能还需修复、哪些浏览器的支持还需增强。

表格还适合比较数据，这在性能测试中尤为常见。例如，测试人员为了比较软件的两个版本之间的性能差异，设计了性能测试表（见表3-4）。该表格测试了两个操作在不同数据集下的耗时。测试完成后，测试人员可以用电子表格软件的“条件格式”功能，使数据单元格中出现反映数值大小的数据条。这使得表格可以直观地显示两个版本的性能差异。此外，他可以生成该表格的数据图（见图3-8），从而可视化地分析数据。

表3-4 性能测试的数据表

		数据集1	数据集2	数据集3
操作1	版本1	10	11	9
	版本2	9	8	7
操作2	版本1	21	22	19
	版本2	17	17	16

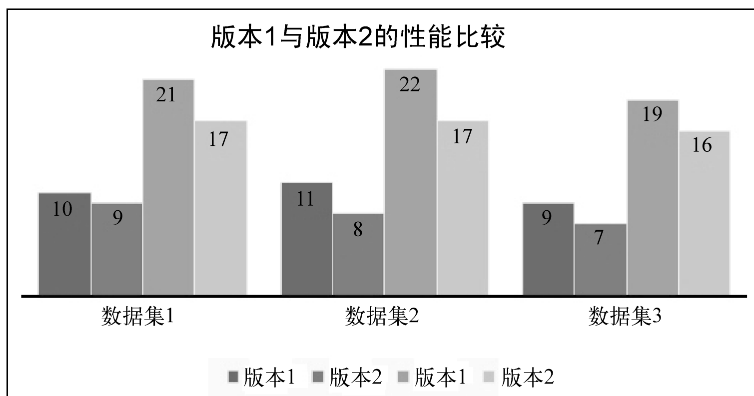


图3-8　性能测试的数据图

此外，表格还常用于表达业务规则。著名的自动化测试框架Fit[①]就使用表格记录测试用例。图3-9是记录在HTML文件中的Fit表格，它以测试用例的形式表达了发放贷款的业务规则[Mugridge05]。其具体含义如下。

- 表头CalculateCredit对应了一个夹具。一个夹具就是一段测试代码，它决定了如何解读表格的数据，并调用被测对象。
- 表格的第二行（标题行），标注了贷款发放的输入值和预期结果，其中months、reliable、balance是输入列，allow credit()和credit limit()是预期结果列。
- 表格的数据行记录了具体的测试用例，例如表格的第三行就明确陈述：当months=14，reliable=true，balance = 5000.00时，allow credit()应该为true，credit limit()应该为1000.00，翻译为自然语言就是“当借贷期限为14个月、客户可以被信赖、结余为5000元时，允许借贷且借贷额度为1000元”。

C:\Users\liangshi\Desk...

CalculateCredit				
months	reliable	balance	allow credit()	credit limit()
14	true	5000.00	true	1000.00
0	true	0.00	false	0.00
24	false	0.00	false	0.00
18	true	6000.00	false	0.00
12	true	5500.00	true	1000.00

图3-9　测试“计算贷款”的Fit表格

① http://fit.c2.com/。

在设计测试时，测试人员与业务人员一起工作。业务分析师用Fit表格编写测试用例，测试人员编写夹具代码，并帮助业务分析师组织测试用例，而这些测试用例阐述了产品的业务规则。在运行测试时，Fit框架读取表格中的数据，调用表格指定的夹具，传入输入值，并检查返回结果。如果返回结果符合预期结果，则测试通过，否则测试失败。Fit产生的测试报告是原表格的复制版，测试通过的预期结果单元格被标记为绿色，测试失败的预期结果单元格被标记为红色。

利用Fit表格，测试人员和业务分析师能够紧密合作，发挥各自的技能优势，实现业务规则的自动化测试。而且，Fit表格是对需求更为清晰的描述方法，能够帮助程序员更好地理解系统、更快地测试、更安全地重构代码[Mugridge05]。所以，Fit并不是一款单纯的测试框架，在许多开发者眼中它是业务分析师、测试人员和程序员的协作工具。

在以上几个例子中，表格既是测试设计的产物，又是测试执行和测试报告的工具。这说明在实际的测试工作中，测试计划、测试笔记和测试报告应该紧密联系。在一些情况下，它们可能来源于同一份文档。

3.2.7　测试指南

测试指南针对特定领域，阐述被测对象的特征和相应的测试策略，帮助测试人员了解基本的领域知识，并实施有针对性的测试。测试人员在工作中积累了许多技能和经验，在进入一个新领域时，他可能需要一些帮助，以便将已有知识应用在新的测试任务中。一份好的测试指南能够让测试人员快速上手，发挥他已有的技能，并避免一些典型的错误。

微软在Windows 8中引入了一种新的应用：Windows 8 应用（俗称“Metro应用”）。为了帮助开发者更好地开发和测试Windows 8 应用，微软Windows团队发布了一篇文章Testing Metro style apps in Windows 8[①]。这篇文章可以看作一篇概括性的测试指南，指出了Windows 8应用的特点和对应的测试策略。其内容结构如下[Sundaram12]。

- **测试应用的激活**：介绍了应用的各种激活方式。
- **测试应用的运行**：这是文章的重点，介绍了多个Windows 8应用的特点和相应的测试要点，包括动态磁贴、通知、协议、旋转、触摸、折叠和全屏视图、不同的屏幕尺寸、数据漫游、传感器、错误处理等。
- **测试应用的挂起**：介绍了应用如何正确地挂起和相应的测试要点。
- **测试应用的恢复**：介绍了应用如何正确地恢复和相应的测试要点。
- **测试应用的终止**：介绍了Windows如何终止应用和相应的测试要点。
- **测试应用在不运行时的行为**：在主程序不运行的情况下，Windows可以通过动态磁贴、通知、后台任务来执行应用的部分代码。该小节介绍了相关知识和测试要点。

① http://blogs.msdn.com/b/windowsappdev/archive/2012/07/12/testing-metro-style-apps-in-windows-8.aspx。

这篇测试指南以应用的状态为核心，介绍了Windows 8应用所特有的知识，并提供了针对性很强的测试策略。读完这篇文章，测试人员可以掌握Windows 8应用的基本测试知识，从而更有效地策划对自己产品的测试。

除了产品之外，一些专项测试也需要编写测试指南。例如，全球化是一些软件的重要需求，它会全面地影响数据存储和界面展示。因此，软件的大多数功能都需要考虑全球化需求，测试小组的每个人都需要考虑全球化测试。但是全球化测试是一个专业性比较强的领域，需要较多的知识积累才能有效实施。为此，测试经理可以安排一名测试人员作为全球化测试的负责人，让他深入研究该领域的编程和测试技术，并撰写一份测试指南。这份指南提供全球化测试的基本知识、测试方法、常见缺陷、常用工具、参考资料等内容。测试小组成员阅读这份文档，并与该负责人交流，从而针对自己负责的功能设计全球化测试。在交流与测试的过程中，负责人可以逐步改进测试指南，使它更符合项目的特点和测试的需要。

3.2.8　测试想法列表

测试想法列表是一种简化的测试指南，列出了一组测试想法，帮助测试人员构思具体的测试。最著名的例子是Quality Tree Software发布的测试启发式方法备忘单①[QualityTree06]。图3-10是该备忘单的局部，提供了一些数据攻击的想法，所针对的数据类型包括路径与文件、时间与日期、数字与计算。该备忘单将测试想法进行了分类，测试人员可以快速找到相关的测试想法。每一个测试想法都用简单的词汇、短语或实例来说明，测试人员能够快速知晓，立即应用。

Data Type Attacks

Paths/Files　**Long Name** (>255 chars) ▪ **Special Characters in Name** (space * ? / \ | < > , . ( ) [ ] { } ; : ‘ “ ! @ # $ % ^ &) ▪ **Non-Existent** ▪ **Already Exists** ▪ **No Space** ▪ **Minimal Space** ▪ **Write-Protected** ▪ **Unavailable** ▪ **Locked** ▪ **On Remote Machine** ▪ **Corrupted**

Time and Date　**Timeouts** ▪ **Time Difference between Machines** ▪ **Crossing Time Zones** ▪ **Leap Days** ▪ **Always Invalid Days (Feb 30, Sept 31)** ▪ **Feb 29 in Non-Leap Years** ▪ **Different Formats** (June 5, 2001; 06/05/2001; 06/05/01; 06-05-01; 6/5/2001 12:34) ▪ **Daylight Savings Changeover** ▪ **Reset Clock Backward or Forward**

Numbers　**0** ▪ **32768** (2^{15}) ▪ **32769** ($2^{15}+1$) ▪ **65536** (2^{16}) ▪ **65537** ($2^{16}+1$) ▪ **2147483648** (2^{31}) ▪ **2147483649** ($2^{31}+1$) ▪ **4294967296** (2^{32}) ▪ **4294967297** ($2^{32}+1$) ▪ **Scientific Notation** (1E-16) ▪ **Negative** ▪ **Floating Point/Decimal** (0.0001) ▪ **With Commas** (1,234,567) ▪ **European Style** (1.234.567,89) ▪ **All the Above in Calculations**

图3-10　测试启发式方法备忘单（局部）

虽然该备忘单不可能穷尽所有的测试想法，但是受到许多测试人员的肯定。其根本原因是它舍弃了所有修饰，用最简洁和直接的方式提供测试灵感，在很短的篇幅内提供了许多有价值的建议。此外，其简洁的形式让测试人员可以随时增加测试想法。随着测试想法的补充，该列表会拥有更多的领域知识，从而更切合当前项目。

① http://www.quality-testing.com/wp-content/uploads/2009/06/testheuristicscheatsheetv1.pdf。

另一个典型的测试想法列表是The Test Eye发布的“37个测试想法来源”[TheTestEye12]。图3-11是根据该文档生成的思维导图，它包含项目、产品、关系人、业务、团队、外部6个方面，共37个测试想法的信息源。测试人员可以依次考虑每个信息源，从中获得测试灵感或行动计划。虽然该测试想法列表比较抽象，难以直接用于测试执行，但是它帮助测试人员从多个角度考虑测试，能够发现原有测试设计的盲点。

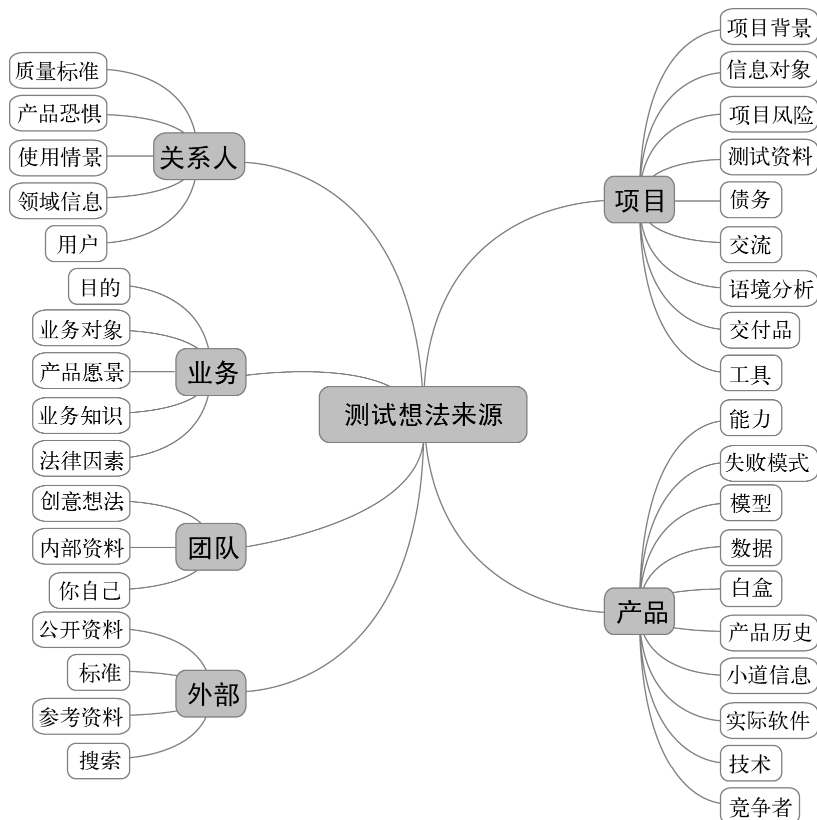


图3-11　37个测试想法来源

此外，测试专家Rikard Edgren发布的免费电子书*The Little Black Book on Test Design*[①]记录了他收集整理的大量测试想法[Edgren11]，也很值得测试人员参考。

3.2.9　质量特性列表

质量特性列表记录了软件需要考虑的质量因素。它帮助测试人员识别出产品需要支持的质量

① http://thetesteye.com/blog/2011/09/the-little-black-book-on-test-design/。

因素，从而为进一步的测试设计提供线索。

质量特性列表的典型例子是The Test Eye发布的软件质量特性集[①][TheTestEye11]。图3-12展示了该文档的结构，其主要质量特性包括：能力、可靠性、可用性、魅力、安全性、性能、IT能力、兼容性、可支持性、可测性、可维护性、可移植性。每个主要质量特性进一步细分为具体的质量特性，限于篇幅，图3-12只展示了能力和性能的具体质量特性。

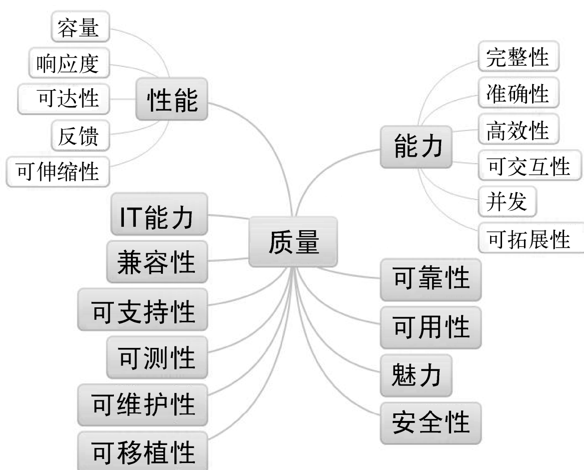


图3-12　软件质量特性集

在使用该质量特性列表时，测试人员需要根据项目情况对其进行修订。可以删去一些与产品无关的质量特性，补充一些与产品相关的质量特性，从而获得一份针对性更强的列表。利用定制的质量特性列表，检查现有的测试设计。如果有某个质量特性没有被仔细测试，就需要设计新的测试策略来覆盖该特性。从这个角度，质量特性列表是一种测试覆盖率检查工具，它帮助测试人员发现测试设计的不足，并提示新的测试策略。

3.2.10　操作文档

测试是一项技术性很强的工作，有些任务可能很复杂，对于新人尤其困难。例如，我参与过一个项目，被测产品是一个由多个网络服务构成的系统。测试小组使用一组虚拟机模板创建虚拟机，在虚拟机上安装不同的服务，然后将多个虚拟机组成测试环境。为了搭建一个测试环境，测试人员需要知晓使用哪些模板、创建何种角色的虚拟机、在特定角色的虚拟机上安装哪些服务、在哪台机器上运行测试、自动化测试的配置文件如何编写等信息。对于一个刚加入项目的测试人员而言，即便他拥有丰富的测试经验，也很难独自搭建一个最典型的测试环境。这阻碍了他的日常工作，令他受挫。

① http://thetesteye.com/posters/TheTestEye_SoftwareQualityCharacteristics.pdf。

为了帮助测试人员完成任务，测试小组可以编写一批操作文档[1]。每个文档针对一个常见的任务，用文字描述、屏幕截图等方式记录操作步骤，使得测试人员可以按照指令完成任务。常见的操作文档涉及安装被测产品、配置测试环境、编写自动化测试、使用测试工具等。测试小组无需为所有任务编写操作文档，只需要针对复杂的任务撰写文档。久而久之，这批文档将成为项目知识库的重要部分，甚至程序员也会时常参考。

随着软硬件的发展，视频录像成为新形式的操作文档。用屏幕录制软件和麦克风录制一个10分钟的操作演示视频，省时省力，其传递知识的效果可能胜过一份用几十分钟编写的文档。有一次，我与一个异地的团队进行测试协作，主要任务是用他们开发的工具测试我的产品。他们提供的是一个10页左右的PowerPoint文档，概述了任务的目标和内容。文档末页链接了一个视频文件。该视频长约5分钟，演示了该工具的用法。我用10分钟便看完了PowerPoint文档和视频，之后就能运用该工具进行测试。可见，多媒体的操作指南使得信息传递更加高效。

3.2.11　检查列表

在完成一项复杂的任务时，人们需要分析并处理多个细节。但是，人脑的记忆力并不可靠，在时间压力下，它很可能会遗漏一些重要的细节，使任务面临失败的风险。为了任务完成的一致性和完整性，人们发明了检查列表，并成功地应用于航空、医疗等关键领域[WikipediaChecklist12]。测试专家Cem Kaner具有法学学位，他指出律师在处理复杂法律问题时会使用检查列表，而软件测试人员也应该用检查列表来指导并追踪测试工作 [Kaner08b]。

检查列表针对一项复杂任务，由一组工作事项组成，每一个事项提醒工作者检查一项细节或完成一项活动。这些工作事项是完成该任务的必要条件，需要工作者认真对待。有时，工作者会根据实际情况跳过一些工作事项，但这是他谨慎考虑后的结论，不是因为遗忘而导致的疏忽。

图3-13是必应词典的安装测试的检查列表。安装测试要检查安装程序可以将必应词典正确地安装在不同的操作系统（Windows 7、Windows 8、Windows Server 2008、Windows Server 2012等）和CPU架构（x86和x64）上。除了检查安装过程以外，测试人员还需要对安装后的程序做一些基本测试，以发现隐藏的安装错误。图3-13的检查列表针对必应词典的主要功能，列出了一些基本的检查点。它不追求详尽的测试（这属于功能测试的范畴），但是覆盖了最基本的功能。它列举了工作项，但没有规定详细的测试用例，这让测试人员不会遗忘重要的测试，又有自主发挥的空间。随着项目的发展，测试人员可以调整检查列表，删除低风险条目，加入新发现的高风险条目。

有些检查列表也可以视作测试想法列表。例如，Michael Hunter的*You Are Not Done Yet: Checklist*[2]针对不同测试对象提出了一组很好的检查列表，覆盖了桌面软件的许多测试任务[Hunter10]。这组

---

① Microsoft Developer Network（MSDN）提供了许多操作文档，例如Quick Start Guide for Manual Testing using Microsoft Test Manager（http://msdn.microsoft.com/en-us/library/vstudio/dd380763.aspx）。

② http://www.thebraidytester.com/downloads/YouAreNotDoneYet.pdf。

检查列表不针对特定的软件，而是提出了一批通用的测试想法，供测试人员做因地制宜地修改。测试人员在完成测试设计或执行之后，可以对照检查列表来发现已有测试尚未覆盖的方面，从而添加新的测试来提高测试覆盖。

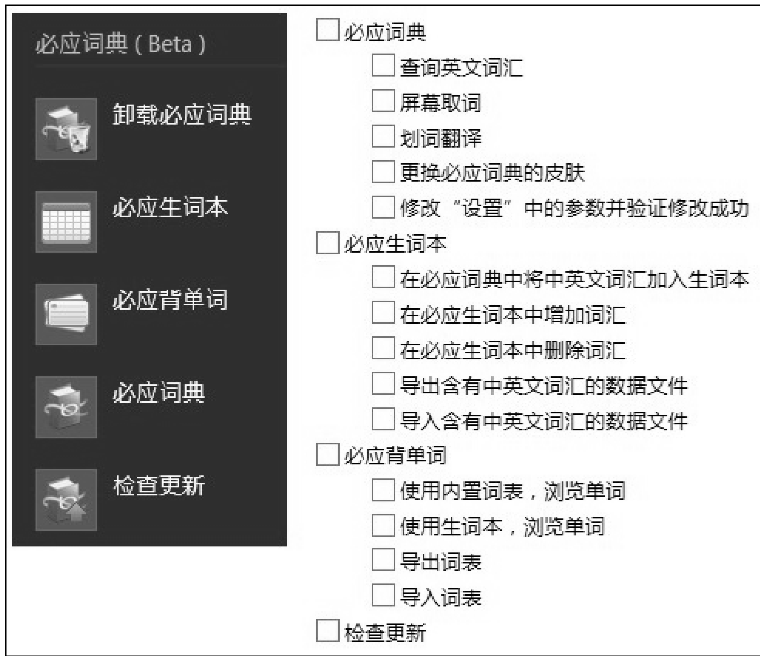


图3-13　必应词典安装测试的检查列表

检查列表是一种提示物，它帮助测试人员在面对复杂的产品时不会遗漏重要的工作事项。构建检查列表的过程就是学习软件并建立模型的过程，是一种有价值的测试学习和测试设计活动，其成果是有结构的检查点或测试想法。在使用检查列表时，测试人员在检查列表（及其背后的测试模型）的指导下，积极地设计并执行测试，充分发挥自身的经验和技能。可见，检查列表让测试学习、设计、执行都是一种高认知工作[Kaner08b]。

3.2.12　缺陷目录

软件缺陷常常体现了编程语言的陷阱、计算平台的不足、程序员能力的欠缺、领域模型的偏差等问题。收集并整理典型缺陷有助于吸取教训、对症下药，是一项有价值的活动。为此，一些软件测试专家整理了缺陷目录（bug taxonomy）供测试人员参考。例如，Cem Kaner等在《计算机软件测试》（*Testing Computer Software*）的附录“常见软件错误”[①]中描述了12类共400多个

① Cem Kaner提供了该附录的全文下载：http://www.testingeducation.org/BBST/testdesign/Kaner_Common_Software_Errors.pdf。

软件缺陷[Kaner99]。之后，Cem Kaner与Giri Vijayaraghavan协作，总结了当时已有的缺陷目录，并讨论了如何更好地使用缺陷目录[Vijayaraghavan03]。

当测试人员获得一份缺陷目录后，他可以从多个角度利用该文档。

- 浏览缺陷目录，标注出产品可能出现的缺陷。利用这些缺陷，评估现有测试设计的质量。如果有某个缺陷可能被漏测，就需要设计新的测试来检查该缺陷。此时，缺陷目录是测试覆盖评估工具，帮助测试人员更周密地测试。
- 将缺陷目录作为“攻击指南”，根据典型缺陷和产品的特点，来做快速测试。
- 提炼出特定领域的缺陷（如安全性缺陷、性能缺陷等），作为测试指南等文档的附录。在评审和使用文档的过程中，这些“常见缺陷”能激发测试灵感，并帮助识别典型问题。
- 分析缺陷目录，找出可能发生的缺陷，然后与程序员交流这些潜在的问题。一方面，程序员可以帮助他理解技术细节、设计测试策略。另一方面，程序员能从中学习到软件或平台的潜在风险，这有助于他提前准备、规避错误。
- 在测试小组内讨论这些典型缺陷，从而帮助测试小组建立团队测试策略，实施有针对性的测试。

有时，测试人员会测试一些新型的应用，在该领域并没有相对完整的缺陷目录。这时，他可以自己归纳该领域的典型缺陷。即便所获得的缺陷列表篇幅不长，也会很有帮助。例如，测试专家Jonathan Kohl所总结的移动应用七宗罪（7 deadly sins of mobile apps）[Kohl12]，就概况了移动应用的典型问题，对于测试设计很有启发性。

- **野心**：应用承诺它做不到的事情。
- **暴食**：应用使用了太多的资源。它占满了内存，拖慢了设备，耗尽了电量，用完了数据流量。
- **贪婪**：应用假设用户总是拥有强劲的网络连接，而不可考虑信号微弱的情况。
- **懒惰**：应用反义迟缓，速度极慢。
- **狂怒**：应用与其他应用不能协作。
- **嫉妒**：应用抄袭其他应用的功能，毫无创新。
- **傲慢**：应用难以使用，期望用户适应它的古怪设计，而不是适应用户。

3.2.13　测程表

为了有效地管理测试，测试经理需要评估测试团队的生产力、当前测试的进度、测试覆盖的范围、已经暴露的风险、测试人员是否需要帮助等因素。一个好的测试流程可以帮助测试经理和测试小组了解这些因素，并实施积极的管理。为了满足软件开发团队对可管理性的要求，Jonathan Bach和James Bach提出了基于测程的测试管理（Session-Based Test Management，SBTM）[Bach00b]。

SBTM将测试过程分解为一组测程，从而提高整个测试项目的可说明性。为此，一个测程包含4个要点：主题、时间盒、可评审的结果和简报[Bach04a]。

1. 主题

主题是一个测程需要完成的任务。该任务拥有具体的目标，可以在90分钟内完成，并提供有价值的简报。主题通常用一段简练的文字描述，其内容可以是测试一个功能、检查一个风险、测试一组用户情景等。以下是两个实例[Bolton07]。

- 为产品DecideRight生成测试覆盖大纲和风险列表。
- 探索产品QuickBuild如何产生一个决策：用户向导功能应该帮助用户使用选项、标准和权重来计算最佳决策。

图3-14是测试专家Elisabeth Hendrickson建议的测程主题模板[Hendrickson13]，其基本结构："使用（资源），探索（特定对象），以求发现（信息）"。

- 资源是测试人员实施探索的工具，它可以是软件工具、测试数据、测试技术、测试对象所依赖的功能的等。
- 对象是需要调查的产品元素，它可以是一个功能、一个模块、一条需求、一个用户情景等。
- 信息是技术调查需要获得的结果，它可以是测试模型、质量属性、潜在风险等。

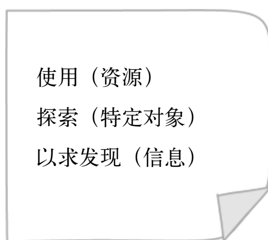


图3-14　测程主题模板

该模版不是编写主题的"最佳实践"，并不适合所有的测程。但是作为思考工具，它可以帮助测试人员思考测程的使命，明确测试的对象、手段和目标。利用该模板，以上实例可以重写为：

- 利用功能漫游，探索产品DecideRight，以生成测试覆盖大纲和风险列表。
- 利用情景测试，探索产品QuickBuild的用户向导功能（包含选项、标准、权重），以检查它能否顺利地产生决策。

主题是测程的指南针。好的主题阐述了探索的对象、要达成的目标、可利用的资源，但不会对具体行动和结果做出不必要的约束。如图3-15所示，从严格的脚本测试到自由式探索的渐变谱系之中，存在许多测试方法，其中主题是偏自由探索的测试方法[Bach10]。在它的左侧是严格脚

本化的测试（即严格按照详细脚本进行测试）、概况的脚本（即没有详细指定所有步骤的测试脚本）、片段的测试用例（即只包含测试要点而省略了其他内容的测试用例），在它的左侧是角色（即测试人员将自己扮演为某类用户）和自由式探索（无需预先准备的测试）。可见，主题的特征是它指定了测试策略，而具体的测试战术则留给测试人员在测程中选择。

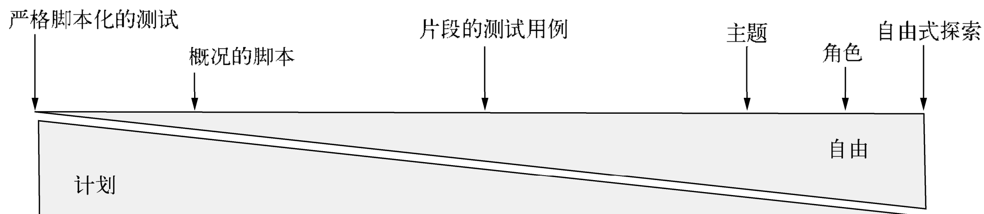


图3-15　主题是偏向自由探索的测试方法

2. 时间盒

时间盒是一段不受打扰的测试时间，其长度一般在60~120分钟，以90分钟较为常见。在此期间，测试人员不回答问题，不回复邮件，不应答即时消息，只专注地实施测试。从工程师的角度来看，时间盒使测试人员排除干扰，全力应对测试的智力挑战。从测试团队的角度来看，固定且专属的时间盒使得测试规划和进度追踪变得更容易。

可评审的结果是测程的产出，常见的形式是测程表，其内容可以包括以下几点。

- 主题
- 测试人员
- 测试所覆盖的区域
- 测试设计和测试发现
- 测试所发现的缺陷
- 测试所发现的问题
- 测试所使用的数据文件
- 测试活动的时间统计：在产品安装、测试设计与执行、缺陷调查与报告、非测试活动中各花费了多少时间

3. 简报

简报通过面对面的交流将测试情况传递给测试经理。在一天的测程结束后，测试人员向测试经理当面汇报测试情况。经理查看测程表，提出一些问题，测试人员解释测试结果，并回答疑问。经理也可以召开小组会议，让测试人员轮流报告当天的测试结果，使测试小组对当前产品的质量形成较完整的认识。

传统上，测程表是一个文本文件，拥有固定的格式。这样，测试经理可以用一个文本处理程

序从多个测程表中提取信息，形成聚合的测试总结报告。测试人员也可以用程序读取测程表，将其中的缺陷记录自动提交到缺陷管理系统。表3-5记录了一个测程表的实例[Bach00b]和相关解释。

表3-5　测程表实例

测程表内容	解　释
主题：分析产品MapMaker的视图菜单的功能，并且报告该区域的潜在风险	测程的主题定义了一个可以在时间盒内完成的任务。该任务将提供有价值的质量信息
区域： OS | Windows 2000 Menu | View Strategy | Function Testing Strategy | Functional Analysis	区域描述了测试范围。本案例通过列举操作系统（Windows 2000）、被测试的菜单（View）、被测试的功能（Function Testing和Functional Analysis）说明了测试覆盖的内容
开始时间：2000/11/11，09:30 AM	
测试人员：Jonthanc Bach	
任务分解： #持续时间 短测程 #测试设计和执行 65分钟 #缺陷调查和报告 25分钟 #测程准备 20分钟 #主题 vs. 机会 100/0	测程表记录了一些任务分解数据，如在测试设计和执行、缺陷调查和汇报、测程准备等活动上花费的时间。这些数据配合简报有助于测试领导估算测试速度、评估测试效率 所谓“机会”是指测试人员在测程中测试了主题以外的功能。在本案例中，“主题 vs. 机会”是100/0，这表示测试人员完全专注于主题，没有测试其他功能
数据文件： FuncTest.mm FuncAnalysis.mm	测程表列举了测试所使用的数据文件，为测试数据复用提供了基础
测试笔记： 我测试了View菜单下的条目，测试重点是地图的缩放行为，测试覆盖了多种地图元素的组合 View： ❑ 欢迎屏幕 ❑ 导航 ❑ 定位器地图 ❑ 图例 ❑ 地图元素 ■ 公路级别 ■ 街道级别 ❑ 机场图	测程表的核心是测试笔记，它简略地叙述了测试故事：为什么测试，如何测试，为什么认为测试是足够好的。本案例介绍了测试策略，并通过功能列表记录了测试覆盖的范围

（续）

测程表内容	解　释
❑ 放大 ❑ 缩小 ❑ 缩放级别 　■ 级别1～14 ❑ 以前的视图	测程表的核心是测试笔记，它简略地叙述了测试故事：为什么测试，如何测试，为什么认为测试是足够好的。本案例介绍了测试策略，并通过功能列表记录了测试覆盖的范围
风险： ❑ 地图元素可能会错误显示。 ❑ 中断重绘可能导致错误显示。 ❑ 光盘可能无法读取。 ❑ 用户可能使用旧版本的光盘。 ❑ 在特定的缩放级别下，产品的某些功能可能出错	测程表记录了风险、缺陷和问题，它们不但是测程的直接产出，还是规划未来测试的参考资料
缺陷： #Bug 1321 当缩放到特定级别（街道名称）时，产品提示插入光盘2。插入光盘2后，产品仍旧提示插入光盘2。 #Bug 1331 快速放大时，街道名称中的结果未能呈现。 #Bug （未录入） 当光驱缓慢或显存较低时，功能不稳定。该缺陷还在调查中	测程表拥有固定的格式，该实例用特殊符号“#”标记了文本处理程序可以提取的数据。测试人员只要遵循简单的格式，就可以产生易于自动分析的测程表。一个简单的文本处理程序能够批量地处理测程表，产生测试报告和图表，并完成自动化任务（如提交缺陷记录到缺陷管理系统）
问题： #问题1 我如何知道哪些细节应该出现在什么缩放级别上？ #问题2 我不确定定位器地图是如何工作的。用户应该如何与其交互？	疑惑是一种测试工具[Kaner01]。测试人员可以用自己的困惑引导测试的方向，设计更多的测试用例来探索这些问题

测程表的本质是测试报告，向团队提供有关产品和测试的信息。分析表3-4不难得出测试报告的基本需求。

- ❑ 测试报告的主要目的是为读者提供所需的信息。例如，测程表的读者是测试经理或测试同伴，因此它的内容比较技术化，常常报告产品和测试的细节以描述测试过程和结果。
- ❑ 报告测试执行者，以明确测试和报告的责任。
- ❑ 报告测试活动的主题，以明确测试的目标和任务。
- ❑ 报告测试的情况，包括测试对象、测试范围、测试策略、测试技术、测试数据等。这有助于报告读者评估测试的深度、广度和质量。
- ❑ 报告测试所发现的缺陷，并阐述哪些功能表现稳定，哪些功能缺陷较多。
- ❑ 报告测试所发现的风险，为项目级别的风险管理提供支持。
- ❑ 根据测试情况，提出需要投入更多测试资源的领域，包括值得进一步调查的问题、值得

深入测试的区域、值得尝试的新测试策略等。

- 提供有关测试过程的信息，为过程改进提供支持。

近年来，出现了更多的SBTM支持工具[Carvalho11]，能够支持多媒体格式的测程表。例如，RapidReporter[①]可以方便地生成CSV和HTML格式的测程表，CSV格式便于自动地提取数据，HTML格式可包含富文本的测试笔记和屏幕截图。

通过聚合测程表收集的数据，测试经理可以评估团队的测试速度。图3-16显示了已执行测程的总时间随日期的变化趋势，这有助于测试经理评估在项目结束前还可以执行多少测程[Bach04a]。如果余下的测试时间不足以完成测试使命，他需要采取措施，以避免项目失败。

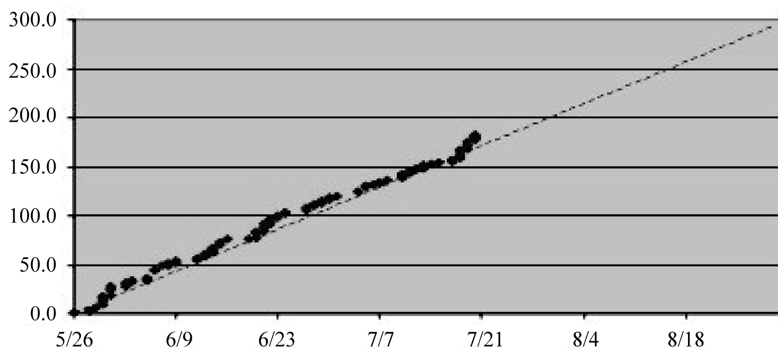


图3-16　已执行测程的总时间随日期的变化趋势

SBTM的本质是动态管理。在项目之初，测试小组对产品还不够了解，测试经理可以安排一些“侦查型”的测程去学习产品的各个区域。例如，上文提到的“为产品DecideRight生成测试覆盖大纲和风险列表”就侦查了产品的主要功能和风险。基于这些测程的测程表和简报，测试领导可以拟定测试项目的总体计划，并大致规划出测程的数目与主题。

在项目过程中，好的测试经理会通过测程表和每日简报，积极发现产品或测试的问题，提出有针对性的解决方案。例如，测试经理老王通过阅读测程表，发现测试人员小张常常花费过多的时间在缺陷调查上。他与小张面谈以了解具体情况。面对面的交流使信息得到高效的传递，并有助于消除统计数据的误导和书面文字的歧义。如果小张喜欢打破砂锅、追根究底，老王会告诉他：在调查缺陷时，可以设定一个最长用时（如10分钟），当时间用尽时，应该停止调查，根据已知情况提交缺陷报告。此外，他还会安排一些有技术挑战的任务给小张，以帮助他增进技术水平。如果小张因不了解产品而花费了过多的调查时间，老王会安排有经验的员工指导小张，或亲自传授一些知识和技能，以帮助他渡过难关。如果是糟糕的可测试性导致了过长的调查时间，老王会和编程小组联系，提出改进意见，以推动产品可测性的提高。

① http://testing.gershon.info/reporter/。

随着产品和项目环境的变化，测试内容与策略也要作相应的调整。测试经理会根据测程的结果来调整下一步的测试计划。他可以新增几个测程，以调查最新发现的风险；他也可以合并几个测程，将省下的时间分配给更重要的测程。这些决策的事实基础就是测程表和简报。

3.2.14　移交文档

当测试人员离开当前项目时，他可以考虑撰写移交文档，帮助接手其工作的同事尽快上手。移交文档的价值在于作为领域专家的测试人员系统性地写出专家知识和经验。与测试设计规约等文档侧重于“前瞻”不同，移交文档更侧重“回顾”与“总结”。它包含如下内容。

- **被测对象概述**：解释被测对象在产品中的作用与地位。
- **文档链接**：指向需求说明、设计文档、测试设计规约等文档的链接，为进一步学习提供素材。
- **基本测试想法**：列举10条最重要的测试想法。当测试设计规约已经详细记录测试设计，移交文档的作用是提供一个入门的台阶，所记录的测试想法保证了基本的测试覆盖，同时让继任者可以立即开始测试。
- **经验与技巧**：阐述测试人员在测试实战中积累的经验与技巧，包括好用的工具、未公开的设置、快速的调试手段等。这些技巧是领域知识的具体表现，能够快速提高继任者的工作效率。
- **已知局限**：介绍被测试对象的局限和缺陷。软件不会是完美的，总存在一些不足和不予修复的缺陷。了解被测对象的问题，能够更全面地理解领域，并减少疑惑。
- **测试自动化**：介绍可用的自动化工具和测试用例。
- **测试数据**：介绍可用的测试数据文件、测试数据库等。

移交文档无需长篇大论，只要指出重点即可，详细内容可以用链接指向相关文档、共享数据、测试数据库、缺陷管理系统等信息源。移交文档不是新员工入职手册，无需介绍众所皆知的测试知识，它应该聚焦于被测对象的领域和特点，介绍现有的测试知识、经验和资产，为接手的同事成为新的领域专家提供垫脚石。

3.3　在测试中发展测试文档

3.2节介绍了一批不同形式的测试文档。在实际工作中，测试人员不需要撰写所有这些文档，也不需要在项目之初完成所有的文档工作。本节将介绍如何在测试过程中发展测试文档。

3.3.1　初始测试文档

在项目之初，测试人员可以参考3.2.1节的介绍来编写一份测试计划草稿，主要确定以下内容。

- 任务：测试人员完成什么工作。
- 策略：用什么方法去完成工作。
- 保障：何时用何种资源完成工作。
- 产品：测试工作产出哪些工作制品。

这份草稿的篇幅可以很短（1~2页），其作用是为测试的技术任务提供框架（策略与保障），为测试组织与协作提供基础（任务和产品）。有了这份草稿，测试人员可以与其他项目成员协作，共同发展测试计划。草稿的好处在于，它提供了讨论的基础，其未定型的状态又鼓励讨论者提出更多的想法。

- 与测试经理讨论测试计划，明确测试任务的优先级、测试进度的安排、工作制品的质量要求。
- 与程序员讨论测试计划，请程序员从实现的角度提出测试想法，并评论现有的测试策略。程序员能提出一些测试人员不易想到的测试要点，这对于改进测试策略很有帮助。
- 邀请几个熟悉的同事，组织一个小型的头脑风暴会议，以产生更多的测试想法。参与者不需要很多，只要彼此信任，能够畅所欲言，就可以在30分钟的会议中产生许多很好的想法。参与者的不同背景和不同视角可以增加测试想法的多样性。

除了与他人协作，测试人员还可以做一些“功课”，以发展测试设计。

- 对新开发的功能做一些基本的测试。如果待开发功能还没有实现，可以对相关功能做一些测试。例如，新功能是在PowerPoint的图片处理中增加几种“图片效果”，测试人员可以对图片和周边功能做一些测试。测试的目的是了解新功能的内容、它的运行环境、与它配合的其他功能等，从而建立整体认识。测试产出可以是一份功能列表，它分析了新功能的具体能力，并标记了潜在风险。
- 从团队内部和互联网上收集测试资料，例如测试指南、测试想法列表、质量特性列表、检查列表、缺陷目录等。对收集到的素材进行裁剪、补充、调整，使它们符合项目语境。该素材整理的过程也是学习被测产品和项目环境的过程，能够帮助测试人员更深入地思考产品和测试任务。
- 积极研究规格说明文档，并与产品经理交流，以理解产品要解决的问题和产品如何解决问题。此外，还会探究隐式规格说明，从而获得更全面的理解。常见的隐式规格说明包括竞争对手的产品、产品家族的其他产品、产品的已发布版本、电子邮件、会议记录、采访记录、用户反馈、第三方评论、技术标准、法律法规、领域专著等。

以上交流和研究并不局限于项目早期。测试人员可以在项目全过程持续地交流，逐渐地展开研究。在工作中，我会用笔记软件记录这些讨论与研究的成果。图3-17是我用OneNote[①]收集整理的工作资料。

① http://office.microsoft.com/en-us/onenote/。

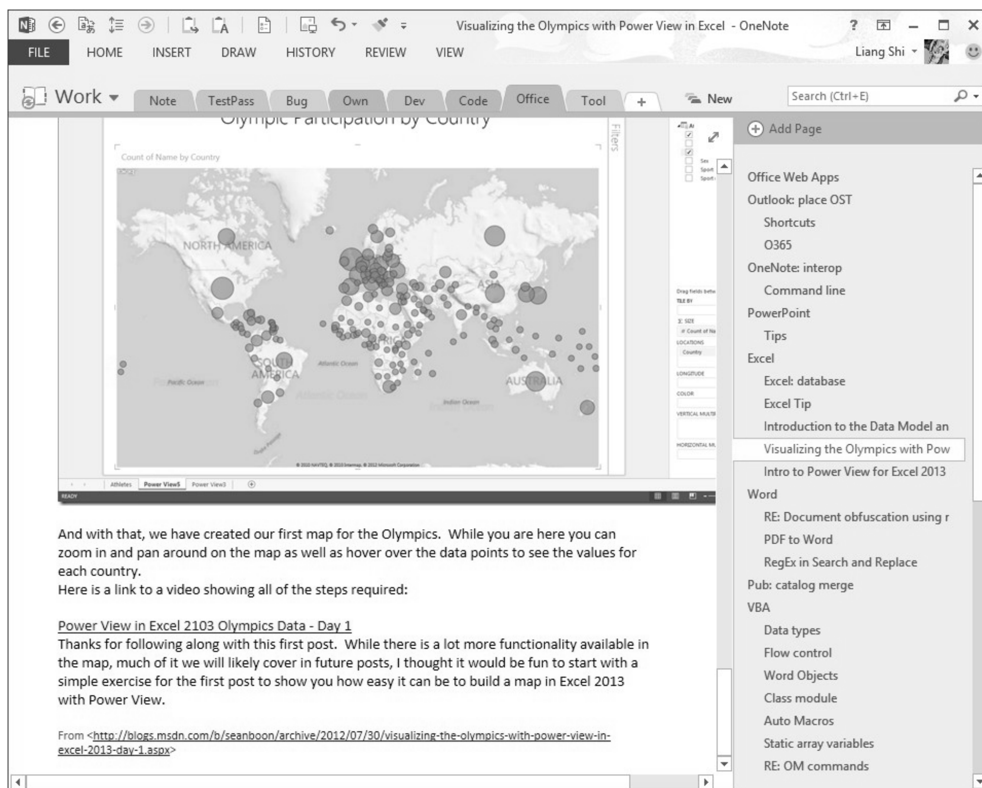


图3-17 用OneNote组织资料

- ❑ 将相同主题的资料放在一个分区中，以实现信息分类。例如，图3-17中OneNote的当前分区是Office，它记录了Microsoft Office相关的技术资料。我还创建了Note、TestPass、Bug等分区，以组织相应类别的信息。
- ❑ 利用页的父子关系表示资料之间的逻辑关系，使得资料组织直观明了。例如图3-17右侧所示的页标签，它通过层次结构进一步对信息分类，便于快速找到PowerPoint、Excel、Word等具体类别的信息。
- ❑ 页可以容纳文档、图片、链接、文字和图表等多种元素，能够将相关主题的内容一网打尽。
- ❑ OneNote提供了全文搜索功能，让资料检索简单且快速。
- ❑ 笔记软件是一个“容器”，容纳了来自方方面面的信息。在某种程度上，它提供了一个工作空间，便于高效地研究多种资料，从而更好地理解产品，更快地设计测试。

当测试想法相对完备后，测试人员可以更新测试计划草稿，以发布可供评审的测试计划初稿。之后，他邀请测试小组、程序员、产品经理等评审测试计划，并根据反馈意见做出修改。评审的一个重要成果是项目团队对测试的任务、策略、保障和产品达成一致意见，使得测试过程可以较顺利地展开。

3.3.2　发展测试文档

测试计划通过评审之后，测试人员需要在测试中持续发展测试文档。如3.1.2节所讨论的，测试人员可以在测试设计、测试执行、测试评估、测试学习等各个阶段发展测试文档。测试是迭代过程，大的循环周期是测试人员测试每一份构建，小的循环周期是测试人员用上一步测试的结果驱动下一步测试的方向。在迭代中，测试文档为新的测试设计提供灵感，又记录了新的测试想法和测试发现。

Cem Kaner等建议测试人员在测试执行时，手边要放一个笔记本，随时记录新的测试想法。他们甚至在硬件资源相对紧张的1999年便提议：更好的做法是，在一台计算机上测试，在另一台计算机上更新测试计划[Kaner99]。我的切身感受是，测试计算机最好能连接两块大屏幕显示器，在一台显示器上测试产品，在另一台显示器上使用笔记软件来发展测试文档（10.1.3节也讨论了多显示器的优点）。如图3-18所示，左侧的屏幕有笔记软件OneNote和思维导图软件FreeMind[①]，OneNote记录了测试笔记（包括测试想法和观察结果），FreeMind展示了质量特性大纲；被测软件“必应词典”和测试工具（Process Explorer[②]和Fiddler2[③]）在右侧的屏幕，它们完成测试执行与监控。

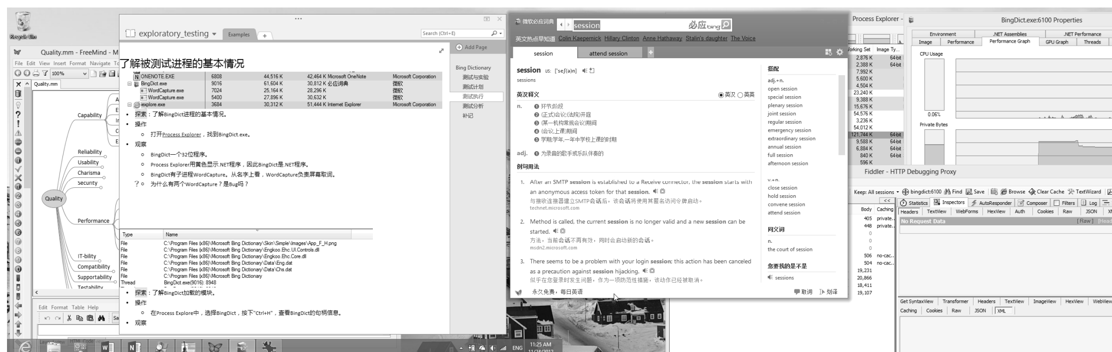


图3-18　在测试的同时更新测试文档图

- 在测试执行时，可以随时查看测试笔记和相关测试文档，用文档中的测试想法去设计具体的测试用例。此外，我会时刻质疑文档的完整性，检查既有测试想法的不足，以激发出更多的测试思路。
- 将测试执行发现的新信息记入测试笔记。主流的笔记软件支持快速地录入、截屏、格式化、标记，允许立即记录测试发现，且没有额外的负担。例如，OneNote支持快速排版，

① http://freemind.sourceforge.net/wiki/index.php/Main_Page。
② http://technet.microsoft.com/en-us/sysinternals/default。
③ http://www.fiddler2.com/fiddler2/。

用键盘快捷键生成符号列表、数字列表、表格和标签，它还支持快捷键Win+S将屏幕截图插入当前笔记页面，从而便捷地产生图文并茂的测试笔记。

- 由于测试笔记记录了即兴的想法、发现的问题、产生的疑惑、捕获的细节，因此不必花精力记住测试过程的点点滴滴。这有利于专注于测试，自由自在地思考与行动。
- 在测试过程中，修改功能列表、思维导图、数据矩阵等测试资料，使它们反映新的知识和测试设计。建议用多种形式的文档记录测试设计和测试发现。测试语境复杂多变，应该选择合适的记录方式，而不是局限于某种特定的方法。我将测试笔记作为核心文档，用它链接其他资料，构成集中组织的测试文档集。
- 经过测试执行的补充，测试笔记所包含的测试设计会更加全面和深入，所记录的技术细节会让测试执行更有针对性。几周之后，当我再次测试相同的功能时，我能够快速上手，而不必苦苦回忆。
- 测试笔记是测试报告和测试评审的原始素材。对其内容稍作整理，就可以形成测试总结报告，让项目团队了解测试的进展。此外，将报告发送给程序员、产品经理和测试伙伴，请他们评审测试设计，能够获得更多的测试想法。

对于在何处增加测试设计的深度，Cem Kaner等提出了6个区域，值得测试人员参考[Kaner99]。

- **最可能发生错误的区域**。如果测试人员知道哪些区域可能存在大量缺陷，就在那里仔细地测试。缺陷往往聚集在一起，应该尽早测试这些高危区域。
- **错误最明显的区域**。从用户的角度看待产品，尝试去发现那些用户最容易发现的错误。这让测试人员专注于对用户最重要的情景和功能。
- **最常使用的程序区域**。测试用户最常使用的功能，以提高软件的稳定性。
- **最有差别的程序区域**。产品想要赢得用户需要提供独一无二的特性，这些特性被称为软件的“卖点”。测试人员需要仔细测试这些区域，因为在卖点上失败将导致竞争失利。
- **最难修正的区域**。许多软件缺陷没有被修正是因为没有时间完成高难度的修复。应该尽早测试难以修正的区域，以便程序员有更多的时间去修复缺陷。
- **测试人员最理解的区域**。通过测试一个熟悉的区域和它的周边区域，测试人员能够快速地测完一个局部，并更好地理解产品。这有助于他更有效地进入下一个区域。

总而言之，测试人员应该在测试中逐步发展测试文档。首先编写一个测试草案，明确任务、策略、保障和产品，并建立起测试设计的框架。然后，通过测试迭代增加测试设计的广度和深度，将重要的设计成果和测试发现纳入测试文档集。

3.4 小结

本章讨论测试文档的编写与维护策略，并介绍了一批具体的测试文档。

- 测试文档是项目团队或测试人员实施测试的工具。

- 测试文档的根本价值在于帮助测试人员更有效地测试。偏离此价值的文档过程将浪费资源、危害项目。
- 在复杂的软件开发中，“测试文档”并不是一份文档，而是一组文档。测试人员要根据项目语境，决定测试文档的类型、内容和形式。
- 测试文档要为读者服务。在许多情况下，测试文档最重要的读者是测试人员自己。
- 用核心文档组织测试素材，构成测试文档集，让多份文档相互支持、彼此参考。
- 首先建立测试设计的框架，然后在测试学习、测试设计、测试执行、测试评估的迭代中，注重实效地发展测试文档。
- 了解不同形式的测试文档，收集不同内容的测试资料，有助于全面思考、灵活应用。

第4章

测试建模

所谓测试建模，就是以测试为目的建立产品模型。建模对象可以是系统、组件、功能等不同粒度的被测目标，建模的结果是一个指导测试设计的模型（简称“测试模型”），它省略了一些与测试无关的信息，强调了值得测试关注的信息。

实际上，所有的测试都基于模型。正如第1章所论述的，软件包含业务、设计、技术等多方面的细节，其复杂度已经超越了人的理解能力。为了实施对质量信息的技术调查，测试人员必需使用一组简化的模型来分析产品和设计测试。即便测试人员没有特意去建立模型，他的测试活动也是基于他对产品的抽象认知，即一些隐式模型。如果他有意识地分析产品和业务，形成针对产品的测试模型，就可以更高效地测试。本章将介绍一些基本的测试模型和建模方法，为测试人员创建针对产品的模型提供基础。

4.1　从组合测试看建模的重要性

本节首先介绍组合测试的基本概念和方法，然后讨论如何根据项目语境来完善组合测试的测试模型，从而说明测试建模的重要性。

4.1.1　组合测试简介

组合测试（combinatorial testing）是一种测试用例生成方法。测试人员将被测试对象抽象为一个受到多个变量影响的系统，其中每个变量的取值是离散且有限的。然后，他使用组合测试工具[①]生成满足特定组合覆盖标准的组合测试用例集（本节简称“测试集”）。

- 两因素组合测试（pairwise combinatorial testing，也称配对测试、全对偶测试）生成的测试集可以覆盖任意两个变量的所有取值组合。在理论上，该用例集可以暴露所有由两个变量共同作用而引发的缺陷。
- 多因素组合测试（n-way combinatorial testing，n>2）生成的测试集可以覆盖任意n个变量

① http://www.pairwise.org/tools.asp列举30余种组合测试工具。

的所有取值组合。在理论上，该用例集可以发现所有由n个因素共同作用引发的缺陷。

在理想情况下，测试人员应该测试所有变量的全部取值组合，即使用全组合测试集。但是，该测试集可能非常庞大，会超出测试的可用资源。为了按时完成测试，测试人员常用组合测试来生成规模较小的测试集。

例如，测试人员实施产品的配置测试，需要测试的软件配置如表4-1示。由该表可知全组合测试需要$2\times3\times4\times2\times3\times3=432$个配置组合。这需要大量的测试时间，可能给项目进度带来风险。根据他的测试经验，配置测试发现的缺陷大多是由一个或两个配置参数所暴露的，只要覆盖任意两个配置参数的取值组合，就能发现大多数配置缺陷。于是，他利用微软的组合测试工具PICT[1]来生成两因素组合测试用例集（本节简称“两因素测试集”）。

首先，他用一个文本文件描述测试模型，即该测试需要考虑哪些变量，每个变量有哪些取值。图4-1是他根据表4-1制作的模型文件，其中一行记录了一个组合测试的变量，“:”之前是变量名，之后是变量的可能取值。该模型文件定义了配置测试的配置需求。

表4-1　配置测试的参数和取值

配置参数	取　　值
PLATFORM	x86, amd64
CPUS	Single, Dual, Quad
RAM	2GB, 4GB, 8GB, 16GB
HDD	SCSI, IDE
OS	WinVista, Win7, Win8
IE	8.0, 9.0, 10.0

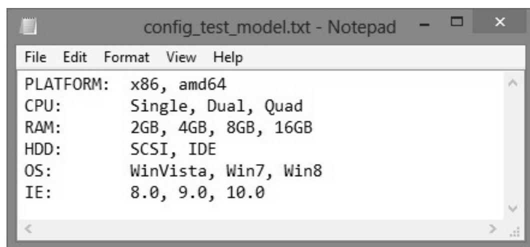


图4-1　PICT的模型文件

然后，他在命令行上调用“pict.exe config_test_model.txt /o:2 > pairwise_cases.txt”[2]，生成如表4-2所示的两因素测试集，其特征是任意两个变量（如PLATFORM-CPU、CPU-HDD、RAM-OS

① http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi。

② 在该命令中，config_test_model.txt是输入的模型文件，/o:2的含义是生成两因素测试集，pairwise_testcase.txt是输出结果。

等）的取值组合都被覆盖。这时，测试用例个数由全组合的432个下降到16个（即全组合的规模是两因素组合的27倍），从而加快了测试进程，且没有显著降低错误发现能力。

表4-2　两因素组合测试用例集

PLATFORM	CPU	RAM	HDD	OS	IE
amd64	Dual	16GB	SCSI	WinVista	8
x86	Single	8GB	IDE	Win7	10
amd64	Quad	8GB	SCSI	Win8	9
x86	Quad	4GB	IDE	WinVista	10
amd64	Single	2GB	IDE	Win8	8
x86	Single	2GB	SCSI	WinVista	9
amd64	Single	4GB	SCSI	Win7	8
x86	Dual	8GB	IDE	Win8	10
x86	Dual	16GB	IDE	Win7	9
amd64	Quad	2GB	SCSI	Win7	10
amd64	Dual	4GB	IDE	Win8	9
x86	Dual	2GB	SCSI	Win7	8
x86	Single	16GB	IDE	Win8	10
x86	Quad	8GB	IDE	WinVista	8
amd64	Quad	16GB	SCSI	WinVista	10

测试人员还可以使用PICT生成模型文件的三因素、四因素、五因素、六因素（即全组合）测试集，其测试用例的个数如图4-2所示。由图中可知，随着因素的提高，测试用例的个数呈非线性增长。在测试实践中，受到测试资源的限制，许多测试人员仅使用两因素组合测试。

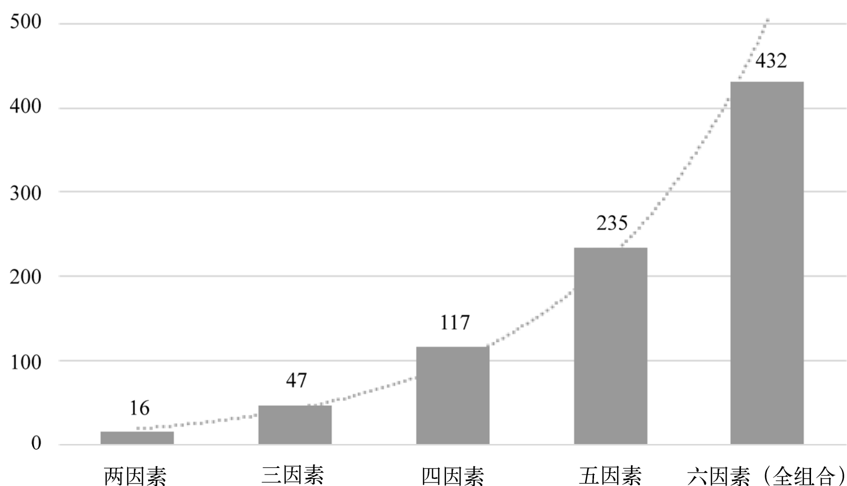


图4-2　测试集的规模随因素个数快速增长

4.1.2　根据语境来完善组合测试的模型

由4.1.1节可知，组合测试针对的问题是没有足够的测试资源来运行所有的测试用例（全组合测试集），提出的解决方法基于一个数学模型和一个假设。

- 所基于的数学模型如公式（4-1）所示，产品的功能被抽象为函数f，产品的输入被抽象为函数的变量$x_1, x_2, \cdots, x_m$，且x_i（$1 \leqslant i \leqslant m$）的可能取值是有限的，产品的输出被抽象为函数的返回值$y_1, y_2, \cdots, y_n$。
- 所依赖的假设是，如果测试覆盖了任意 t 个（$2 \leqslant t < m$）输入变量的取值组合，那么该测试可以发现函数 f 的大部分错误。对于两因素组合测试而言，其假设是如果测试覆盖了任意两个输入变量的取值组合，那么它可以发现产品的大部分错误。

$$y_1, y_2, \cdots, y_n = f(x_1, x_2, \cdots, x_m) \qquad (4\text{-}1)$$

以上数学模型和假设就构成了组合测试的测试模型。它非常简单且抽象，可以广泛地“应用”于不同的测试活动。但是，正因为它过于简单且抽象，测试人员需要根据产品特征来完善该模型，才能使它“适用”于当前的测试活动。本节将介绍一些组合测试可能遇到的问题，并讨论如何完善测试模型来解决问题。

问题1：组合测试的理论没有讨论如何建立其数学模型

组合测试的测试模型很简单，但是组合测试的文献很少讨论如何根据产品来构建该模型[Bach04b]。然而，测试人员在实施组合测试时，必须解决以下问题。

- 确定变量$x_1, x_2, \cdots, x_m$，即确定被测产品的输入变量。测试产品的一个功能，要考虑许多因素。如图4-3所示，PowerPoint可以对图片实施“阴影”特效。在测试该功能时，测试人员除了考虑图4-3所呈现的7个设置项，还需要考虑图片类型、图片像素、图片是否透明、图片是否具有其他特效、图片是否与其他元素“组合”、图片的输出类型等变量。测试人员不需要通过一次测试去覆盖所有这些变量，但是他应该谨慎选择当前的测试需要覆盖哪些变量，并计划更多的测试以覆盖余下的变量。如果他的测试设计遗漏了重要的变量，那么组合测试也会错失重要的测试用例。
- 确定每个变量x_i的取值集合。
- 表4-3是图4-3所列7个设置项的可能取值个数。即便测试人员只考虑这7个变量，表4-3的两因素测试集也包含海量的测试数据。为了获得小规模的两因素测试集，测试人员常常用等价类划分技术来获得每个设置项的“典型取值”。如果他的等价类划分遗漏了一个重要的等价类，那么组合测试也会错失重要的测试数据。James Bach对两因素组合测试的有效性进行了实验。他发现他构造的两因素测试集错过了许多缺陷（缺陷发现率不到50%），缺陷遗漏的主要原因是该测试集没有包含可以暴露错误的特定取值[James04b]。也就是说，如果变量x_i的取值集合不够好，组合测试的效果就会大打折扣。

- 确定检查方法，以判断返回值$y_1, y_2, \cdots, y_n$是否正确。测试人员需要构造测试先知以检查产品的输出。不严谨的检查有可能遗漏已经暴露的缺陷。



图4-3　PowerPoint的图片阴影功能

表4-3　设置项的取值个数

设置项	可能取值的个数
预设	24
颜色	16 581 375
透明度	100
大小	200
模糊	100
角度	360
距离	200

遗憾的是，现有的组合测试研究并没有很好地回答以上问题。这体现了组合测试面临的根本性挑战：纯粹的数学模型远离产品的业务模型和代码的实现模型，不能直接应用于测试。为此，测试人员需要完善组合模型，使之符合业务模型和实现模型。在此过程中，测试人员需要深入地研究业务领域、产品需求、代码实现、测试技术等内容，付出持续的努力且无捷径可走。

问题2：组合测试可能会错过最重要的取值组合

图4-4是Word 2013“高级”设置的一部分。为了测试Word在不同设置下的行为，可以将一个单选框视为一个变量（取值为“启用”或“关闭”），对它们生成两因素测试集。该测试集看似较好地覆盖了不同变量的组合，但是很可能没有覆盖Word的默认设置。不幸的是，很多用户从不修改默认配置，测试集没有覆盖最常用、最重要的取值组合。这揭示了组合测试的一个风险：组合测试的数学模型可能不适用于被测软件。如果测试人员不仔细分析产品，只依赖组合测试工具，他可能错过重要的测试用例。

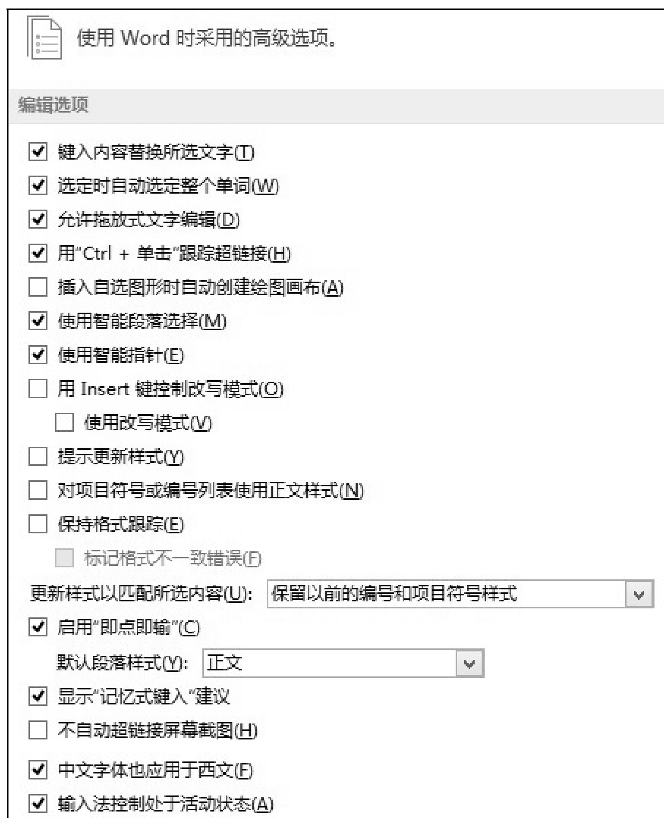


图4-4　Word的"高级"设置

为了避免漏测，测试人员应该利用领域知识和测试技能，发掘出一批必须测试的取值组合。考虑到Word用户的使用模式，测试人员可以参考如下测试策略。

(1) 将默认设置对应的取值组合加入测试集。

(2) 对于默认设置，每次修改其中一个单选框，获得一批取值组合。将它们加入测试集。

(3) 考虑到大多数用户不修改默认设置或只修改1个设置，该测试用例集已经"足够好"，可以将其付诸测试。

图4-4中一共有19个单选框。如果不考虑单选框之间的依赖关系，以上策略将产生20条测试用例。如果使用PICT生成这19个变量的两因素测试集，该集合包含10条测试用例。虽然手工生成的测试用例要多于PICT的结果，但是它们更贴近用户所使用的设置，是更好的测试集。

如果测试人员认为覆盖任意两个变量的取值组合很重要，他可以先构造面向业务的测试集，然后将其作为"种子"，利用PICT生成两因素测试集。例如，命令"pict.exe model.txt /e:seedrows.txt /o:2"所生成的测试用例集就包含种子文件（seedrows.txt）所指定的取值组合。

问题3：组合测试的数学模型没有描述变量之间的约束关系

在组合测试的数学模型中，每个变量的取值是相互独立的，即x_i的取值不会影响x_j的取值。但是，大多数被测对象的变量之间存在约束关系。以表4-1所列的配置参数为例，当变量PLATFORM的取值是x86时，变量RAM的取值就不能是8GB或16GB，因为x86 CPU最大只支持4GB内存。即便测试人员将组合（PLATFORM: x86，RAM: 8GB）作为负面测试用例，也是不可行的，因为x86的主板根本插入不了8GB的内存。

可见，如果不考虑变量之间的约束关系，测试集将包含大量的无效测试用例。因为组合测试工具生成的每一条测试用例对于因素覆盖都是必要的，仅仅删除无效测试用例，会导致最终的测试集不能实现两因素或多因素组合覆盖。对于此类被测对象，测试人员应该明确定义变量的约束关系，让组合测试工具根据约束来生成有效的测试集。

例如，在PICT的模型文件中，加入如下的约束语句，就可以定义变量之间的约束关系。

```
IF [PLATFORM] = "x86" THEN [RAM] NOT IN ("8GB", "16GB");
IF [OS] = "Win8" THEN [IE] >= 10.0;
```

第一条约束是，当CPU是x86时，内存不能是8GB或16GB。第二条约束是，当操作系统是Windows 8时，IE的版本号要大于或等于10.0。当PICT读取模型文件时，它会解析约束规则，并将其应用于测试用例生成过程。生成的测试集既满足组合覆盖标准，又不包含无效取值组合。

在大多数情况下，无业务知识的数学模型不会是高效的测试模型。定义约束是根据产品的业务模型来改进组合测试模型。测试人员需要仔细分析业务规则，发掘出应该加入组合测试模型的约束，使组合测试工具有更丰富的信息来处理输入变量。

问题4：组合测试用例可能被“卫哨”语句过滤

许多软件用卫哨语句来“过滤”无效的输入。例如，在如下代码中，if语句就是一个卫哨，它检查函数的输入，一旦发现A<=1就退出函数func(…)。

```
int func(int A, int B, int C)
{
    if (A <= 1) return ERROR;
    ...
}
```

如果测试人员没有读过代码，或没有仔细分析规格说明，他可能会制定如下的模型文件。在该模型中，A的取值是 0, 1, 2。

```
A: 0, 1, 2
B: 0, 1, 2
C: 0, 1, 2
```

利用上述模型，PICT所生成的测试用例集包含10条测试用例。

```
A       B       C
1       2       0
2       0       2
0       1       0
2       2       1
0       0       1
1       1       2
2       0       0
0       2       2
1       0       1
2       1       1
```

在这10条测试用例中，有6条测试用例因为A<=1会被`if`语句过滤掉。只有4条测试用例，能够执行到if之后的语句。这意味着只有40%的B和C的取值组合被真正地覆盖。这个例子表明，如果没有考虑卫哨语句对执行流的影响，测试集将不能达成两因素或多因素覆盖的目标 。

面对此类问题，测试人员要仔细阅读规格说明或源代码，发掘会导致执行流跳转的“非法”取值。他可以将非法取值从模型中排除，将变量的取值置于正常执行流的范围。例如，对于函数`func`，他可以将模型文件定义为：

```
A: 2, 3, 4
B: 0, 1, 2
C: 0, 1, 2
```

在生成测试集之后，再加入一条的测试用例（A: 1，B: 1，C: 1）以覆盖“非法”取值。原模型生成的测试用例可以“通过”卫哨语句，覆盖因素A、B、C的两两取值组合；附加的测试用例可以覆盖卫哨语句的“过滤”功能。

另一种方法是在PICT的模型中用特殊符号“~”标记出“非法”取值。例如，在如下模型中，参数A的取值1被标记为非法。

```
A: ~1, 2, 3
B:  0, 1, 2
C:  0, 1, 2
```

PICT会保证所有有效值的取值组合都会被覆盖，此外任意非法值与有效值的组合也会被覆盖。以上模型将生成如下测试用例集。

```
A       B       C
2       2       0
2       1       2
3       0       0
2       0       1
3       1       0
3       0       2
3       1       1
3       2       1
```

```
2      2      2
~1     1      0
~1     0      1
~1     2      2
```

如果很清楚被测对象的实现逻辑，使用第一种方法可以生成规模较小的测试用例集，因为它几乎不考虑非法值与有效值的组合。如果只是从规格说明中了解到程序可能存在卫哨语句，那么用第二种方法可以生成更“安全”的测试集。这两种方法都说明测试人员需要利用软件的实现模型来丰富组合测试的数学模型，否则测试集很可能无法达到组合覆盖的目标。

问题5：组合测试可能没有提供足够的测试覆盖

在软件测试过程中，最先找到的缺陷往往处于程序的“主干”上，典型的执行流就可以覆盖相应的路径或状态；被遗漏的缺陷常常位于程序的“末枝”，程序执行需要满足特殊条件才能覆盖相应的语句或状态。这是软件测试的经验之谈，也提示两因素组合测试也许不能发现隐藏在“末枝”中的缺陷。一些路径需要多个变量的特定取值组合才能被覆盖，然而两因素组合测试不能保证测试集可以覆盖这些组合。因此，在测试资源允许的情况下，引入多因素组合覆盖有可能进一步提高错误发现率。例如，在命令行上执行“pict.exe model.txt /o:3”，可以生成三因素测试集。

然而，并不是所有的变量都会协同工作。例如，在如下函数compute中，输入变量$x1$和$x2$会影响输出变量$y1$，输入变量$x3$和$x4$会影响输出变量$y2$。变量$x1$与$x2$协同工作，变量$x3$与$x4$协同工作，但是$(x1, x2)$和$(x3, x4)$之间并没有联系。因此，实用的测试策略是周密测试$x1$和$x2$的取值组合、以及$x3$和$x4$的取值组合，简单测试$x1$、$x2$、$x3$和$x4$的取值组合。

```
void compute(int x1, int x2, int x3, int x4 int* y1, int* y2)
{
    *y1 = foo(x1, x2);
    *y2 = bar(x3, x4);
}
```

虽然函数compute是一个极度简化的例子，但是它反映了复杂产品的特征：它拥有多个输入变量和多个输出变量，一个输入变量不会影响到所有的输出变量，一个输出变量也不会受所有的输入变量影响。测试人员可以阅读规格说明，分析程序代码，咨询程序员，从而知晓每个输出变量受哪些输入变量影响。他将相互协作的输入变量纳入小组，高强度测试小组内部的变量组合，低强度测试跨小组的变量组合。

对于PICT，测试人员可以在模型文件中定义子模型。例如，他在模型文件中增加如下语句，以定义两个子模型{PLATFORM, CPU, RAM, HDD}和{OS, IE}。

```
{ PLATFORM, CPU, RAM, HDD } @ 3
{ OS, IE } @ 2
```

然后，他用命令“pict.exe model.txt /o:1”生成测试集。该测试集对PLATFORM、CPU、RAM和HDD实施三因素组合覆盖，对OS和IE实施两因素组合覆盖。

这个例子再次表明，测试人员需要根据软件的业务规则和代码实现来完善组合测试模型。合理的子模型使得测试人员可以严密地测试相关变量的取值组合，又不会显著提高测试集的规模。

问题6：全组合测试可能是更好的测试策略

组合测试的初衷是减少测试用例的数量，使测试可以在较短的时间内完成。随着硬件性能的发展，自动化测试可以在短时间内执行成千上万的测试用例。而且，云计算技术允许多台计算机并行地执行计算，从而进一步提高了测试执行的速度。这使得测试人员可以利用软硬件的强大性能来执行海量的测试用例。在这种情况下，将测试模型替换为全组合测试可能更好。

例如，表4-1的配置测试的全组合测试需要432条测试用例[①]。假设测试人员用虚拟机完成配置测试，创建一台满足配置要求的虚拟机需要1.5小时，运行自动化功能测试需要1.5小时。那么完成432个配置的测试，需要$432 \times 3 = 1296$小时。如果测试人员向测试实验室申请20台虚拟机，完成全部测试需要$1296 \div 20 = 64.8$小时。这意味着他在周五下午4点提交测试，周一上午9点就可以查看所有测试结果。可见，合理的自动化测试可以充分利用空闲的计算资源，让测试人员既能享用愉快的周末，又能完成繁重的测试任务。

另一种接近全组合测试的方法是每次回归测试都使用全新的测试集。有些团队将组合测试用例集加入回归测试集，在项目过程中复地执行。这时，不妨每次生成新的组合测试用例集，一方面满足两因素或多因素覆盖的要求，另一方面扩大测试对程序状态空间的覆盖。如果每次都使用相同的测试用例，测试集可能只是反复执行相同的路径，覆盖相同的状态空间，也许不能发现隐藏的缺陷。如果每次都用新的测试用例，随着回归次数的增长，测试执行可以执行更多的路径，覆盖更广大的状态空间，发现隐藏缺陷的概率也会提高。

在PICT中，参数“/r[:N]”为测试用例生成引入随机种子（N是作为随机种子的整数），以生成不同的测试用例集。例如，如下命令行指令用当前日期作为种子，生成测试用例集。

```
set seed=%date:~4,2%%date:~7,2%%date:~10,4%
pict.exe model.txt /r:%seed%
```

第一条语句从当前日期中获得年、月、日信息[②]，放入变量seed中。第二条语句以变量seed为种子，生成测试用例。于是，测试集的具体内容随日期变化，在保证两因素覆盖的前提下，逐渐覆盖了更多的程序状态。

行文至此，本节已经讨论了实施组合测试的6个问题和相应的解决方法。从中不难看出测试建模对于高效软件测试的重要性。

---

① 由问题3可知，这432条测试用例包含一些“无效”用例，实际可执行的测试用例数要小于432条。为了简化讨论，这里仍旧使用432条测试用例来估算测试时间。

② %date的格式随Windows系统设置而变化，该语句只适用于特定Windows系统。

- 通用的测试技术（如组合测试）往往基于“产品中立”的测试模型，虽然可以“广泛”应用，但是需要测试人员根据产品来完善其测试模型，才能“有效”应用。
- 完善测试模型需要参考产品的业务领域，例如根据领域知识抽取恰当的变量、确定变量的取值、定义变量之间的约束关系等。
- 发展测试模型还需要参考产品的代码实现，例如根据“卫哨”语句调整变量的取值集合、对变量分组以实施子模型的组合测试等。
- 综合运用多种技术和模型能够提高测试效率，例如第6个问题的解决方法就综合利用了组合测试、自动化测试、随机测试和虚拟化技术。
- 测试建模是一个演进的过程。测试人员很难在项目之初就知晓所有细节，所以测试模型的初稿往往是有缺点的。随着项目的发展，测试模型的不足会在测试迭代中逐渐暴露。负责任的测试人员会针对这些问题，调整测试模型，让它“与时俱进”。

最后，我向读者推荐James Bach和Patrick J. Schroeder的文章Pairwise Testing: A Best Practice That Isn't [Bach04b]。这篇文章不但深入分析了组合测试，给出了许多很好的建议，还展示了测试专家研究测试技术的思路和方法。该文指出职业的测试人员应该运用批判性思维来研究他所使用的每项技术，只有透彻地了解技术的优点和缺点，才能根据语境合理地使用技术。

4.1.3 测试建模的基本点

上节讨论了如何完善组合测试的测试模型，来解决实施组合测试遇到的问题。从中不难看出，合理的测试模型可以准确地描述产品，使测试更有针对性，且避免了测试遗漏和浪费。

为了获得好的测试模型，测试人员需要从多个角度考察产品，重要的切入点包括业务领域、软件实现和项目环境。

- **业务领域**：测试建模的对象是被测产品，而构建产品的使命是解决业务问题，因此恰当的模型应该切合产品的业务领域。测试人员可以研究需求文档、客户调查、相关产品、领域专著等资料，也可以与客户代表、产品经理等相关人员讨论，从而获得测试建模的必要信息。
- **软件实现**：好的测试模型具有很强的针对性，能够清晰地描绘出被测产品的特征，这要求测试人员分析产品元素，了解代码实现。具体而言，测试人员可以参考4.2.1节介绍的HTSM之“产品元素”，从结构、功能、数据、接口、平台、操作、时间等角度分析软件实现。他需要学习产品的系统架构、代码结构、数据处理、操作接口等实现细节，还需要学习产品的运行环境、开发平台等背景知识。
- **项目环境**：构建测试模型的目的是为了实施更好的测试，好的测试模型不一定复杂，但一定实用——能够切合当前项目环境，并充分利用项目资源。为此，测试人员可以参考HTSM之“项目环境”，从开发者关系、测试小组、测试设备、测试工具、项目进度等方面考察测试模型的可行性。例如，为期1周的测试和为期4周的测试需要不同的测试模型。

前者的测试模型较简单，会聚焦于高价值的功能和高风险的领域；后者的测试模型更丰富，侧重于测试策略的多样性，并覆盖更多的产品细节。

在研究和思考的过程中，测试人员需要记录测试模型。一方面，测试建模是一个持续的过程，测试人员会根据产品的变化来调整测试模型，记录当前的模型为模型演化提供了基础。另一方面，测试模型应该在测试小组中分享，让更多的人检查模型、提供意见、使用模型能令它更好地发展。具体而言，模型的表达方式可分成形式化模型和非形式化模型两种。

- **形式化模型**可以被工具软件读取，以自动生成测试数据或执行测试。例如，PICT的模型文件所记录的就是一个形式化模型，PICT可以根据它生成测试用例集。本质上，PICT提供了一个建模语言，测试人员使用该语言可以定义被测对象的测试模型，让人和计算机都可以理解建模结果。与PICT相似，许多测试工具提供了测试领域专属语言，让测试人员记录测试模型，并根据模型执行自动化测试。
- **非形式化模型**的表现形式包括自然语言表达的文字、列表、草图等。它们记录了测试人员的思考，为接下来的测试设计提供信息。例如，图3-3的功能列表就是PowerPoint图片功能的非形式化测试模型。它列举了图片的主要功能，提供了测试覆盖的目标，为进一步的测试设计建立了可扩展的框架。

测试建模是一个研究的过程，测试模型是研究的成果。不过，模型并不是最终的目标，它只是辅助测试的工具，其价值只取决于发现缺陷的效率。为了优化在模型上所投入的资源，测试人员应该让实际的测试需求来推动模型的发展。可行的策略是构建一个简单的模型，用它指导测试设计，并根据测试反馈来改进它。在测试迭代中，模型会逐渐成长，日趋成熟。

4.2　常用测试建模方法

随着软件测试的发展，测试建模方法层出不穷，本书无法一一介绍。本节将讨论几个常用的测试模型，为更高级的测试模型奠定基础。

4.2.1　启发式测试策略模型

根据产品的风险设计测试是一种常见的测试设计思路。在复杂的现实世界，产品面临的风险多种多样，只有全面考虑、周密测试才能避免风险暴露导致的严重后果。因此，测试人员需要一个相对完整、可以定制、容易扩展的风险列表或参考模型，来帮助他们发现产品风险。启发式测试策略模型（Heuristic Test Strategy Model，HTSM）是测试专家James Bach提出的一个结构化的、可定制的参考模型，从测试技术、产品元素、项目过程、质量标准等多个角度启发测试设计[Bach12]。

图4-5是HTSM的概要描述，测试人员利用质量标准、项目环境、产品元素，指导测试技术的选择与应用，并产生观察到的质量。

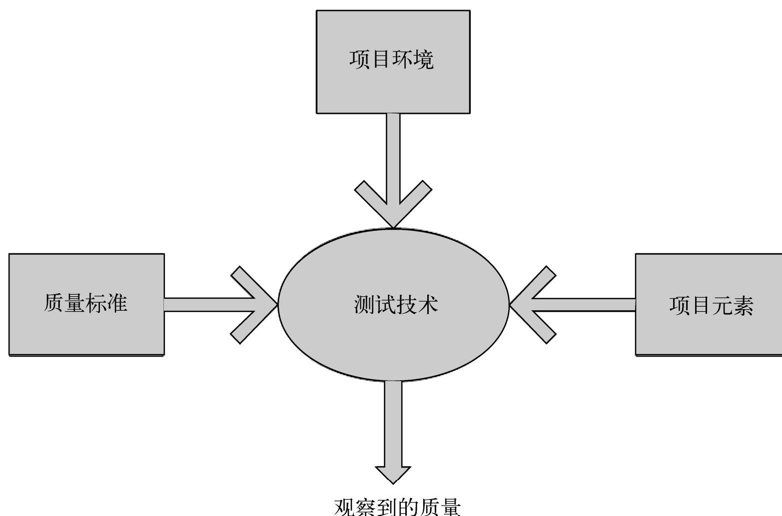


图4-5 启发式测试策略模型

HTSM具有层次结构，其顶层元素（质量标准、项目环境、产品元素、测试技术）可以分解为次层元素，而次层元素可进一步分解为第三层元素。本文只概要介绍次层元素，更多的细节请参考James Bach的文档[①]。

测试技术：生成测试的策略。有效地选择和实施测试技术，需要综合分析项目环境、产品元素和质量标准。可选的测试技术如下。

- 功能测试：测试软件的能力。
- 域测试：测试软件所处理的数据。
- 压力测试：用极限行为和数据压迫软件。
- 流测试：测试软件的操作顺序。
- 情景测试：用有说服力的场景来测试软件。
- 声明测试：测试需求规格、广告宣传等文档的声明。
- 用户测试：邀请用户参与测试。
- 风险测试：猜测软件可能发生的错误，然后寻找它。
- 自动测试：用工具和自动化技术来多角度地检查软件。

项目环境：资源、约束和其他影响测试的项目元素。测试总是受到项目环境的约束。在某个团队运转良好的策略不一定适合另一个相似的团队，以往富有成效的方法未必适应当前的项目。有经验的测试人员会根据当前语境，选择合适的测试实践。项目环境包括如下几个方面。

- 使命：测试人员和其服务对象需要就测试人员的任务达成一致意见。

① http://www.satisfice.com/tools/htsm.pdf。

- 信息：测试所需的有关产品和项目信息。
- 开发者关系：如何与开发者协作以加速开发。
- 测试团队：利用团队的力量支持测试。
- 设备与工具：可利用的硬件、软件和文档等资源。
- 进度：项目实施的流程。
- 测试条目：测试范围和重点。
- 交付品：测试人员的产出。

产品元素：需要测试的对象，包括如下几个方面（2.2.2节从测试覆盖的角度详细讨论了产品元素）。

- 结构：产品的物理元素，如代码、硬件、配置文件、数据文件等。
- 功能：产品的功能。
- 数据：产品所操作的数据。
- 接口：产品所使用的或暴露出的接口。
- 平台：产品所依赖的外部元素。
- 操作：使用产品的方式。
- 时间：影响产品的时间因素。

质量标准：产品需要考虑的质量特性，包括如下几个方面。

- 能力：产品能否完成期望的功能？
- 可靠性：产品能否在期望的条件下稳定的工作？
- 可用性：真实用户能否顺利地使用产品？
- 魅力：产品是否光彩夺目？
- 安全性：产品能否抵御恶意攻击？
- 可伸缩性：产品能否自如地使用软硬件资源？
- 兼容性：产品能否与外部组件和配置协同工作？
- 性能：产品的速度和响应如何？
- 可安装性：产品是否易于安装？
- 面向研发团队的特性：研发团队能否方便地编写、测试和修改软件？

由以上介绍可知，HTSM由一组指导性词语组成，它们构成一个概念框架，让测试人员从高层抽象到底层细节对产品和测试进行思考。这些指导性词汇是测试的指南，其作用不是教导如何具体测试，而是启发测试人员的思维，发掘测试对象和测试策略。

图4-6摘录自James Bach的测试培训教材[Bach11]，体现了HTSM对于测试设计的意义。

- 测试设计以风险驱动。测试人员分析质量标准、项目环境和产品元素中的风险，设计有针对性的测试策略。

- 在测试设计时，质量标准启发测试先知，项目环境启发测试过程，产品元素启发测试覆盖，观察到的质量启发测试报告。
- 对于测试，HTSM强调测试策略的多样性，平衡代价和收益，利用启发式方法充分发挥测试人员的技能。

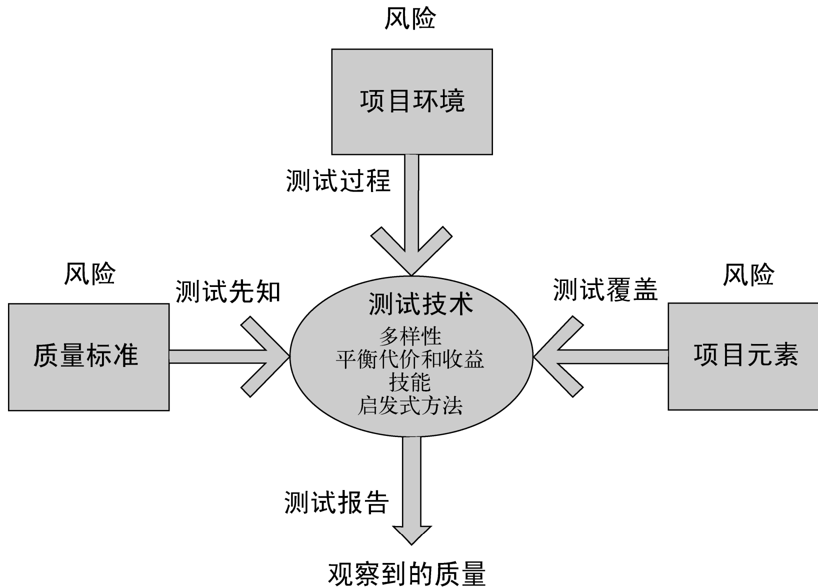


图4-6　HTSM中的测试先知、测试过程、测试覆盖和测试报告

因为HTSM是一个框架，测试人员可以用它对测试方法分类，从而更好地理解这些测试方法的着眼点。例如，表4-4记录了一些测试文档在HTSM中的位置，对于识别这些文档的作用和关系有一定帮助。

表4-4　测试文档在HTSM中的概念

测试文档	在HTSM中的概念
Google ACC之属性	HTSM → 质量标准 → 魅力
Google ACC之部件	HTSM → 产品元素 → 结构 HTSM → 产品元素 → 功能
Google ACC 之能力	HTSM → 质量标准 → 能力
功能列表	HTSM → 产品元素 → 功能 HTSM → 测试技术 → 功能测试
测试想法列表	HTSM → 测试技术
质量属性列表	HTSM → 质量标准
缺陷目录	HTSM → 质量标准
测程表	HTSM → 项目环境 → 交付物 HTSM → 观察到的质量

在定制化之前，HTSM对测试人员的帮助很小，因为此时的HTSM是“James Bach的模型”，而不是符合当前语境的模型。HTSM是通用的模型，虽然能够普遍应用，但是不能快速、高效地指导具体的测试工作。测试人员需要将其“本地化”，才能发挥其威力。

Cem Kaner用思维导图[①]定制自己的HTSM [Kaner11]。如图4-7所示，他将HTSM作为中心，将质量标准、测试技术、项目环境、产品元素作为主干。



图4-7　思维导图形式的HTSM

在分支上，Cem Kaner添加了他觉得重要的节点。例如，在图4-8中，他在产品元素下增加了优点节点和时间节点（其中时间节点来自测试专家Michael Bolton的建议），以满足他的工作需要。

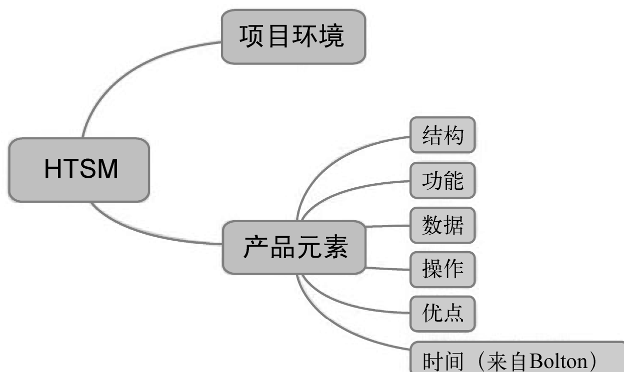


图4-8　通过增加节点来定制HTSM

恰如Cem Kaner所说，“大多数严肃对待此模型的人都会定制它以符合自己的需要”，测试人员应该修改HTSM，以获得符合项目语境的模型。定制HTSM是理解并掌握HTSM的过程。与大多数方法一样，测试人员需要修改它，加入自己的风格和元素，才能真正掌握它。以下是一些常见的修改手段。

- **增加节点**：增加与当前项目相关的测试技术、测试想法、测试对象和任何测试人员认为有价值的元素。
- **删减节点**：忽略一些与项目或任务无关的元素。
- **增加标记、注释、链接等元素**：标记可以突显重要的元素，注释可以增加更多的细节，链接可以指向更详细的信息源。

① 测试专家Michael Larsen在XMind.net上提供了他制作的HTSM思维导图（http://www.xmind.net/share/mkltesthead/），为测试人员制作自己的HTSM提供了很好的范例。

定制HTSM也是应用HTSM的过程。测试人员遵循HTSM的结构，深入地思考产品、项目与测试，添加自己的想法、评论、标记和启发式问题。这本身就是极好的测试学习过程。作为学习的结果，定制后的HTSM为测试设计提供了坚实的基础。在测试过程中，测试人员会接触新信息，学习新知识。他会持续地将新知识补充到HTSM中，以迭代地优化测试模型。从这个角度，HTSM既是测试想法的源头，又是测试过程的产出。在测试设计时，测试人员可以逐个检查HTSM中的元素（指导性词语），阅读相关标记、注释和链接，以启发测试思路。他可以自问：

- 该元素与当前测试任务相关吗？
- 针对该元素，产品有什么风险？可能会有什么缺陷？
- 通过什么测试可以发现这些缺陷？
- 依据当前的进度和资源，如何实施这些测试？

另一种更有威力的方法是综合HTSM中的多个元素，开发测试策略。当程序员用单元测试检查了组件，测试人员需要在系统层面检查产品。此时，产品的缺陷往往存在于组件的交互和复杂的流程。综合产品的多个方面，开发多样的测试才能够更好地体现测试人员的价值。一些有帮助的启发式问题包括以下几点。

- 该元素与哪些元素相关？
- 元素的组合有没有揭示出新的风险？
- 如何设计测试，以同时测试这些元素？
- 能否让来自元素A的信息帮助元素B的测试？

通过思考这些问题，测试人员可以设计出强有力的测试策略。例如，3.2.4节给出了一个复杂的情景测试，用于回归测试PowerPoint的图片功能。为了方便读者阅读，在此将该测试复述如下。

(1) 新建一个PowerPoint文档。

(2) 向文档中插入图片。

- 覆盖所有支持的图片格式。
- 覆盖典型的图片尺寸。
- 覆盖来自单反相机的大型图片。

(3) 操作文档中的图片。

- 覆盖“图片工具”下的所有命令。
- 用一个命令修改图片。
- 用多个命令修改图片。
- 保持一些图片不被修改。

(4) 应用文档中的图片。

- 将图片与其他元素组合使用。
- 覆盖文本框、形状、SmartArt、图表、日期与时间等元素。

(5) 将文档中的元素导出为图片。

- 覆盖所有可以被输出的元素：页面、图片、形状、SmartArt、图表等。
- 覆盖所有支持的图片格式。
- 覆盖相关的用户界面命令：上下文菜单（即右键弹出菜单）、Backstage（点击“文件”→“导出” →“ 改变文件类型”）、复制并粘贴到画图板。
- 覆盖相关的VBA对象模型。

(6) 打印该文档。

- 打印到黑白打印机。
- 打印到彩色打印机。
- 打印到PDF文档。
- 打印到XPS文档。

(7) 保存该文档并重新打开。

- 另存为所有支持的格式。
- 用PowerPoint打开生成的文档。
- 用旧版本PowerPoint打开生成的文档。

分析以上测试步骤，不难看出该测试涵盖了“HTSM→ 产品元素”分支的所有元素。

- **结构**：测试覆盖了位于powerpnt.exe（PowerPoint的可执行程序）、mso.dll（提供图片功能的动态链接库）等文件的代码。
- **功能**：测试覆盖了创建文件、导入图片、修改图片、应用图片（将图片与其他元素组合使用）、导出图片、打印文件、读写文件等功能。
- **数据**：测试覆盖了不同的导入图片类型、导入图片尺寸、导出图片类型。此外，使用“图片工具”的命令修改图片也产生了新的图片数据。
- **接口**：测试覆盖了用户界面和编程接口（VBA对象模型）。
- **平台**：测试覆盖率多种打印机类型，包括彩色打印机、黑白打印机、虚拟打印机（PDF和XPS）。
- **操作**：测试用不同的命令修改图片，用不同的命令序列修改图片。
- **时间**：测试覆盖了插入日期与时间、打印日期与时间、读写文件中的日期与时间。

由这个例子不难看出，构建模型是简化和聚焦的过程，应用模型则是扩展和发散的过程。

- 在创建模型时，测试人员将复杂的项目和软件抽象为相对简单的元素和关联，将注意力

集中在他认为重要的因素上，而省略其他细节。一个精炼的模型易于理解、易于应用，而一个过于庞杂的模型则让人抓不到重点，反而不利于测试实战。

- 在使用模型时，测试人员用模型激发测试灵感，通过发散思维构思出多种多样的测试设计。模型是知识和经验的浓缩，其简要的形式是思考的起点，而不是思考的边界。测试人员需要充分利用模型蕴含的知识去产生尽可能多的测试想法，以扩大测试覆盖。

除了综合HTSM中一个分支的元素，测试人员还可以综合不同分支的元素。例如Google ACC模型就覆盖了多个分支的元素：属性对应“HTSM→质量标准→魅力”，部件对应“HTSM→产品元素→结构”和“HTSM→产品元素→功能”，能力对应“HTSM→质量标准→能力”。用多个角度考察软件，能够开发出强力测试策略，达到事半功倍的效果。

总之，HTSM是一个由测试指导词组成的测试模型，能够在整个测试过程提供帮助。在制订测试计划初稿时，它可以帮助测试人员完整地思考产品，从而产生系统性的测试计划。在测试过程中，它可以帮助测试人员组合测试想法、深入探索产品，以开发出强有力的测试策略。在回归测试中，它可以帮助测试人员确定测试范围，制定测试方案。

4.2.2　输入与输出模型

输入与输出模型（简称IO模型）是最基本的测试模型，它将被测对象（功能、模块、系统等）视为一个整体，分析并列举该对象的输入变量和输出变量。在大多数测试中，测试人员都会有意或无意地使用IO模型，因为它反映了测试的基本需求：掌握输入变量可以控制或影响被测对象的行为，掌握输出变量可以观察行为的后果，从而检查其是否正确。

图4-9是Cem Kaner和Doug Hoffman描绘的被测系统的输入和输出[Kaner08c]，体现了构建相对完整的IO模型需要考虑多个因素。

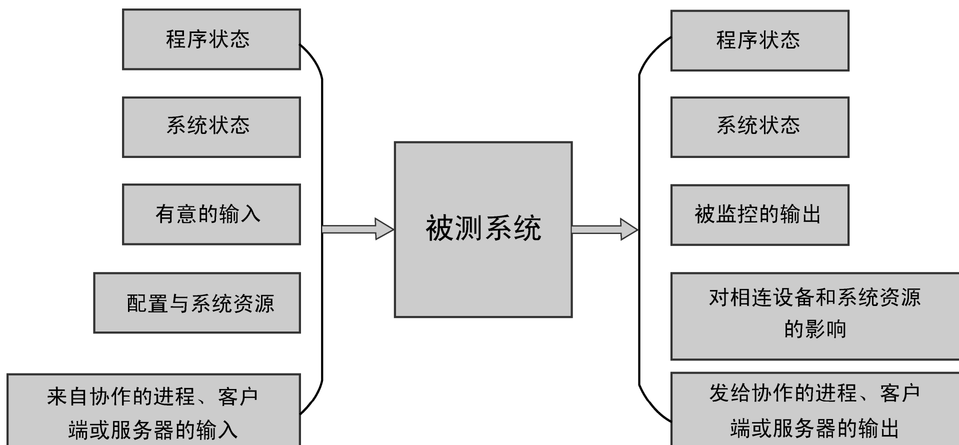


图4-9　被测系统的输入与输出

在输入方面，值得考虑的因素包括以下几点。

- **有意的输入**：为了完成某项任务，用户有意识地操控软件所产生的输入。
- **无意的输入**：用户无意间产生的输入。对此，测试专家Jonathan Kohl分享过一个故事[Kohl12]。某个平板电脑上的软件在测试时很稳定，但在演示时总是出错。经过反复测试，Kohl发现了错误原因。演示时，第一位演示者会将平板递给第二位演示者。在交接的瞬间，两个人都握着平板电脑，这导致屏幕上产生多个触点信号，而且移动所产生的惯性使加速感应器发出信号。该软件不能很好地处理多个并发的信号，于是出错。该案例说明，测试人员需要仔细观察他人和自己的行为，发现认知的盲点，才能找到一些被忽视的重要输入。
- **程序状态**：程序自身的状态，通常包含内存中的数据和程序依赖的外部数据。例如，许多Web应用会从数据库中加载数据，也会把数据更新写回数据库，而这些数据值很可能决定了Web应用的行为。
- **系统状态**：程序所依赖的运行平台的状态。例如，当用户切换Windows 8应用时，操作系统会向被换出的应用发出“挂起”事件，向换入的应用发出“恢复”事件。对于一款应用而言，它要快速响应“挂起”事件，在系统规定的5秒内保存必要的自身状态，它还要正确响应“恢复”事件，来加载已保存的状态，提供流畅的用户体验。对此，测试人员应该构造多个“挂起”和“恢复”事件的序列，以检查软件的相关功能。
- **配置与系统资源**：影响软件行为的配置和平台资源。例如，软件执行某个操作时需要大量的内存。在一台4GB内存的机器上，该操作较容易失败，在一台16GB内存的机器上，该操作就不容易失败。然而，内存资源被同一台计算机上的进程所共享。如果其他进程占据了大部分内存，该操作仍旧会失败。
- **来自协作的进程、客户端或服务器的输入**。例如，对于必应词典，来自必应服务器的单词解释就是很重要的输入数据。

在输出方面，值得考虑的因素包括以下几点。

- **被监控的输出**：被用户观察到的软件输出。
- **未被监控的输出**：没有被监控或没有被观察到的输出。例如，必应词典可以将一个单词“添加到生词本”，该操作会将一条记录写入一个本地的数据库。对于不了解实现细节的用户或测试人员，该输出几乎是“不可见”的。如果测试人员发现了该输出变量，他很容易产生一批测试想法，如测试数据库丢失、数据库损坏、数据库容量耗尽等情况。
- **程序状态**：程序自身的状态，通常包含内存中的数据和程序依赖的外部数据。例如，Web应用会将用户提交的数据写入数据库。测试人员需要检查其输出的数据是否正确、是否被正确地写入数据库。如果所提及的数据包含错误甚至恶意代码，测试人员要检查Web应用不会写入错误数据或破坏数据库。
- **系统状态**：软件运行会修改运行平台的状态，这可能对平台和平台之上的其他进程产生

影响。例如，软件为了完成计算会临时性地锁住某个文件，此时其他尝试访问该文件的进程都会遭遇访问失败。

- **对相连设备和系统资源的影响**。我开发过一个测试程序，它整晚运行测试用例。部署后的隔天，我发现运行该测试程序的计算机失去响应。重启机器后，我对测试程序进行了调试，发现它存在一个内存泄漏的错误。短时间运行该程序，不会产生严重的后果，但是整晚运行会耗尽计算机的可用内存，导致计算机失去响应。
- **发给协作的进程、客户端或服务器的输出**。例如，必应词典在接受用户输入的单词后，会构造一个HTTP请求，将它发送给必应服务器。该HTTP请求可视为它对用户输入的输出，测试人员需要检查该输出是否正确、构造并发送请求的性能是否达到预期等内容。

由以上讨论可知，为了建立完整的IO模型，测试人员需要从多个角度考察被测对象和相关系统。为此，他可以研究项目文档，咨询产品经理，请教程序员，并通过测试来建立IO模型。测试实际的产品，获取第一手的资料，很可能发现一些其他渠道不能获得的信息。

3.2.13节介绍了基于测程的测试管理（Session-Based Test Management，SBTM）。许多测试小组并不使用SBTM组织测试活动，但这并不妨碍测试人员用测程来管理自己的时间。在SBTM中，测程有4个特点：主题、时间盒、可评审的结果和简报[Bach04a]。当测试人员自己运用测程时，通常不需要做简报，于是主题、时间盒和可评审的结果是他考虑的重点。

- 主题是一个具体的测试任务，它可以在90分钟的时间盒内完成，并提供有价值的结果。一个值得参考的主题模板是："使用（资源），探索（特定对象），以求发现（信息）"[Hendrickson13]。对于构建IO模型，主题可以是"利用Fiddler等网络工具，分析必应词典与服务器的通信，以了解必应词典与服务器的输入输出关系"、"利用Process Monitor等调试工具，探索必应词典，以了解它对操作系统的读写操作"等。
- 时间盒是一段不受打扰的测试时间，一般为60~120分钟。测试人员可以根据主题确定合适的时长。在此期间，测试人员专注于工作，不进行与测程无关的活动。
- 可评审的结果是测程的产出。对于构建IO模型，测程的产出就是被测对象的IO模型，它可以是一份列表，记录了所发现的输入变量和输出变量。测试人员可以将IO模型记入测试笔记，为今后的测试提供参考，也可以将它分享给测试小组，使知识得以传播。

面对一个复杂的被测对象，测试人员可以安排一个测程去产生它的IO模型。如果测试对象非常复杂，他可以安排一组测程去生成其IO模型的不同部分，最终获得完整的模型。此类测程通常使用一些漫游测试方法（漫游测试将在5.4节讨论），来探索软件的各个方面。

测程通常要使用一些软件工具。由图4-9所列的输入和输出因素可知，构建IO模型需要分析软件状态、它与操作系统的交互、它与其他软件和服务的交互。不使用工具去查看软件和交互的内部细节，很难实施完整的分析。在Windows平台上，我常使用表4-5所列的软件工具。

表4-5　Windows平台的常用测试分析工具

工　　具	简　　介
Windbg[①]	Windbg可以调试Windows应用和Windows操作系统。我常使用它观察原生程序和托管程序的状态[②]
Process Explorer[③]	Process Explorer可以提供任意进程的详细信息，包括程序路径、命令行参数、性能指标（CPU、内存、IO、GPU、.NET）、句柄、模块、线程和环境变量等
Process Monitor[④]	Process Monitor可以监视并记录一组进程对注册表、文件系统和网络的访问。Process Explorer提供了进程的当前状态，Process Monitor则提供了进程一段时间的运行轨迹，综合使用可以了解进程与进程、进程与操作系统之间的互动[⑤]
Fiddler[⑥]	Fiddler是一个Web调试代理，能够记录并实时修改计算机与网络服务器之间的HTTP(S)通信。我常使用它观察软件与服务器的交互
Windows Performance Toolkit[⑦]	Windows Performance Toolkit是Windows 8 SDK中的性能分析工具，可以详细地记录一组进程的性能指标、调用栈、调用耗时等信息，对于理解进程性能、进程间互动、进程与操作系统间互动很有帮助[⑧]
SQL Server Management Studio[⑨]	SQL Server Management Studio 是SQL Server数据库的管理工具。我常使用它查看被测应用的数据库，以理解数据组织方式和数据状态

在测程中，测试人员可以参考HTSM的“产品元素”，从结构、功能、数据、接口、平台、操作、时间等多方面来发掘变量。

- **结构**：搜索软件所使用的文件。例如，使用Process Explorer查看软件打开了哪些文件句柄，使用Process Monitor查看软件曾经访问过哪些文件。
- **功能**：遍历用户界面，搜索隐藏的功能，分析功能蕴含的变量。
- **数据**：检查软件可以接纳、拥有、处理、输出哪些数据，尝试不同的数据类型和数据范围。例如，使用Fiddler查看软件与其他软件的通信数据，使用SQL Server Management Studio查看软件对数据库的读写。
- **接口**：发掘软件提供的接口和软件使用的接口。例如，有些软件会注册一些操作系统事件的处理函数，当操作系统触发事件时，相应的处理函数会被执行。测试人员要发现这

① http://msdn.microsoft.com/en-us/windows/hardware/gg463009.aspx。

② 关于Windbg的更多信息可以参考*Inside Windows Debugging*（http://book.douban.com/subject/10575139/）、《Windows高级调试》（http://book.douban.com/subject/3781532/）、《.NET高级调试》（http://book.douban.com/subject/5346548/）和《Windows用户态程序高效排错》（http://www.cnblogs.com/lixiong/archive/2010/02/11/1667516.html）。

③ http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx。

④ http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx。

⑤ 关于Process Explorer和Process Monitor的更多信息可以参考工具作者Mark Russinovich所著的*Windows Sysinternals Administrator's Reference*（http://book.douban.com/subject/3794381/）。

⑥ http://www.fiddler2.com/fiddler2/。

⑦ http://msdn.microsoft.com/en-us/windows/desktop/hh852363.aspx。

⑧ *Inside Windows Debugging*（http://book.douban.com/subject/10575139/）详细介绍了Windows Performance Toolkit的前身xperf，对于理解Windows Performance Toolkit也很有帮助。

⑨ http://www.microsoft.com/sqlserver/en/us/default.aspx。

些事件和处理函数。

- **平台**：探索影响软件的外部元素，包括操作系统、操作系统上的其他进程（例如网络防火墙可能会影响网络程序的行为）、与软件协作的其他服务等。
- **操作**：检查不同的操作步骤、使用方法、配置参数是否产生了不同的结果。考虑用户的使用情景，发掘其中的关键因素。
- **时间**：搜索与时间相关的变量。
  - **搜索由时间点触发的事件**。例如，用户在智能手机上设置了闹钟提醒。在预设的时间点，手机操作系统会弹出提示，该提示对于当前通话或当前应用有何影响？
  - **搜索由时间段或时限触发的事件**。常见的例子是"超时"设置会中断当前处理的业务。
  - **搜索周期性的事件**。例如每天、每月、每年会执行的操作。考虑这些操作是否会导致意外的情况，例如一些财会软件会在每日零时结算前一天的业务。如果某项业务的处理跨越零时，会不会产生问题？
  - **检查是否存在跨时区的业务，考虑这些业务有哪些关键变量**。
  - **考虑用户和其他软件的输入速度**。有些软件不能处理快速的输入，会导致数据丢失。
  - **探索最长的计算过程**。这些漫长的计算过程会伴随着资源泄漏吗？如果遭遇中断或异常，它们可以恢复并继续计算吗？

有些变量对应了用户界面控件，较容易发现；有些变量不存在"可视的"界面元素，容易被忽视。以下是一些常见的"隐藏"变量，供读者参考。

- **操作系统事件**。操作系统会向软件发送各种消息，软件需要合理地处理。例如，Windows在关机时，会向应用发出WM_SHUTDOWN消息。大多数软件会忽略该消息，少数软件会根据该消息做一些清理工作（如保存数据、清除临时文件等）。测试人员需要向产品经理和程序员咨询，并做一些实验，以发现软件需要处理的操作系统事件。
- **设备事件**。在智能手机等移动设备上，硬件传感器（如触屏、陀螺仪、加速感应器、光线感应器、GPS等）会发出大量的信号。测试人员需要检查移动应用正确地处理了必要的信号，忽略了无关的信号。
- **中断事件**。用户在使用软件时，操作系统或其他软件可能中断当前情景。例如，用户在使用智能手机应用时，一个拨入的电话会中断当前应用。再例如，用户一边走一边使用移动应用，操作系统突然提示附近有可用的Wi-Fi热点，该通知也中断了当前应用的流程。测试人员需要调查当前平台可能触发哪些中断事件，并将它们用于测试。
- **用户输入方式**。随着软硬件的发展，用户输入方式也日趋多样：物理键盘输入、屏幕键盘输入、鼠标点击、鼠标拖曳、鼠标手势、触摸、触摸拖动、触摸手势、语音输入、手写输入等。测试人员需要发现当前平台可以接收的输入方式，并用于测试。
- **数据类型**。例如，对于图像处理软件，图片类型是重要的变量，PNG、JPG、GIF等不同类型的图片可能会产生不同的结果。

- **数据大小**。大尺寸的数据可能会导致性能问题或计算错误，因此数据大小是一个重要的变量。例如，用户拍摄了50张数码相片，每张相片的文件大小是5MB。图片处理软件可以轻松处理任何一张相片，但是同时处理50张相片时却遭遇内存不足的错误。以此类推，一张250MB的图片很可能导致相似的错误。
- **相对位置**。例如，在队列头部、尾部、中部插入一个新的元素，往往会执行不同的代码，而错误可能隐藏在某段特定的代码中。测试人员要去发现此类由相对位置引发的特定操作。
- **容量**。测试人员需要考虑“容器”的容量，典型的测试值是0个元素、1个元素、n个元素（n是容器可容纳的最大元素个数），例如空的队列、只有1个元素的队列、容量耗尽的队列、空的图文框、包含1个嵌套图文框的图文框、包括9级嵌套的图文框、不弹出确认对话框、弹出1个确认对话框、弹出9个确认对话框。这些变化的数值暗示了软件的不同处理方式，值得测试人员进一步探索。
- **本地化和国际化**。许多操作系统允许用户更改系统的地区，这通常意味着字符集、时间格式、货币格式等基本数据格式的改变。许多软件允许用户更改界面语言，这通常意味着界面字符串的长度、方向、格式的改变，有些软件还会加载新语言的字典文件。测试人员需要检查软件能够处理这些基本数据的变化。
- **时间**。测试人员需要考察特殊的时间点、特殊的时间长度、周期性的事件、涉及多个时区的业务、输入的速度、计算的耗时等。

构建IO模型有助于测试人员更好地理解被测对象，更自如地操控，更全面地观察，更好地设计测试。通常，发现重要的变量，相应的测试想法就会自然浮现。反之，未发现重要的变量，测试设计就会存在漏洞，有可能错过重要的缺陷。

4.2.3　系统生态图

在测试一个较复杂的软件系统时，测试人员需要分析系统的内部结构、外部依赖和访问接口。测试专家Elisabeth Hendrickson建议测试人员绘制一幅图来描述系统的生态环境，它综合了传统的语境图和部署图，可以帮助测试人员系统性地探索整个系统[Hendrickson13]。我将这样的图称为系统生态图。

- 语境图描述系统如何与外部环境联系，展示了系统的用户、接口和外部依赖（如数据源、协作进程、支撑服务等）。
- 部署图描述组成系统的子系统或模块，包括数据库、服务进程、可执行程序等。

图4-10是一张我手绘的系统生态图，描述了一个我测试过的报表系统。

- **从语境图的角度，图4-10的虚线框描述了报表系统的边界**。系统分成两部分，左侧部分是一个Silverlight应用，实际用户通过该Silverlight应用访问报表数据，右侧部分是系统的后台，由系统管理员控制。除了系统边界和用户，图的右上角还呈现了系统所依赖的外

部数据源。

- **从部署图的角度，图4-10描述了系统的组成部分**。这是一个“客户端—服务器”架构的系统。在客户端，一个运行在IE中的Silverlight应用从服务端获得数据，并绘制报表。在服务端，一个同步工具将外部数据源的数据转储在报表数据库中。一个Windows服务将读取数据库，并提供基于WCF技术的数据访问服务。一个IIS网站调用该数据访问服务，把获得的报表数据传递给客户端的Silverlight应用。

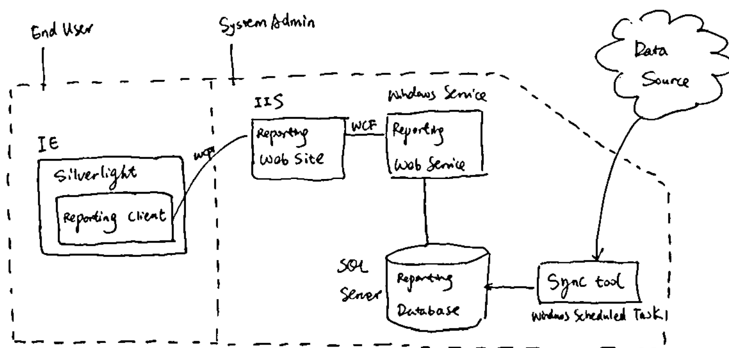


图4-10　一个报表系统的系统生态图

制作系统生态图的时机一般是在系统设计完成的时候。测试人员可以将该图作为评审设计规格说明的“课后作业”，通过绘制测试视角的模型来理解系统设计。他可以从实际用户接触的组件开始，逐渐画出整个系统，也可以从最重要的模块开始，逐渐衍生出其他模块。最初的系统生态图无需追求细致，只要涵盖整个系统即可。测试人员可以随着项目进展对它进行修订。

关于绘制方式，我倾向于用纸笔绘制系统生态图，因为手绘有几个优势。第一，手工绘制的速度超过使用作图软件，我只需几分钟的时间就可以画出图4-10。第二，手工绘制灵活性高，可以方便地画出多种图元，随意地添加标记和注释。第三，手绘让我专注于图的内容，而不是作图软件的功能，有助于更好地思考。第四，我可以将图纸展示给同事，以分享建模成果。在讨论过程中，我们可以快速地将新的想法补充到图上。纸和笔提供了一个自然的协作平台。

手工绘制的主要缺点在于不易存档。为此，我准备了一个文件夹，专门存放手工绘制的各种图纸。如果需要长期保存，我会用智能手机拍摄图纸的照片，将照片保存在电子的测试笔记中。以我的经验，大部分图纸服务于短期任务，无需电子存档，只有少量的重要图纸需要电子化。

手工绘制的另一个缺点是不易于修改。对于细节的修改尚可以在原图上涂改，但是大规模的修订就只能重绘。对此，我有两个应对方法。第一，对于重大修改，我会重新绘制模型。绘制一

幅图只要几分钟的时间，开销很低。更重要的是，重绘不是描摹原图，而是重新思考产品、建构更好模型的过程。舍弃原有模型表达，花一些时间去构思新的形式，往往会显著提升模型的质量。第二，对于复杂、经常改动、或需要正式评审的图形，我会先手工作图，当模型比较稳定后，再使用软件工具绘图。

在获得系统生态图之后，测试人员可以用它指导测试设计。一种测试设计思路是关注系统的边界，检查系统的使用者和外部依赖对系统的影响。对于图4-10，测试人员可以考虑如下方面。

- **针对同步工具从外部数据源导入数据**。常见的测试想法包括：如果数据源暂时不可访问，会如何？如果数据源提供空的数据集，会如何？如果数据源提供海量的数据，会如何？如果数据源提供不符合预期格式的数据，会如何？如果数据源提供语义错误的数据，会如何？
- **针对实际用户访问的Silverlight应用**。常见的测试想法包括：Silverlight应用对用户的计算机有什么要求？ Silverlight应用的错误会不会危害用户的计算机？用户能否绕开权限控制，看到他不应该看到的报表？
- **针对客户端（Silverlight应用）和服务端（网站、Windows服务、数据库）的数据传输**。常见的测试想法包括：如果客户端提交错误的数据，服务端如何处理？如果服务端提供错误的报表，客户端如何处理？如果多个用户同时访问，大量的并发客户端会不会导致性能问题？如果服务端暂时不可访问或反应迟缓，客户端如何向用户报告？

以上所列只是可行测试想法的一小部分。测试人员可以深入分析系统特点，产生更多的测试想法。其中，一个有帮助的思考方法是**条件分析**，即不停地询问“如果发生……，会如何？”通过构思可能发生的各种情况，测试人员能够检查系统对于复杂情景的应对能力。

对于系统生态图，另一种测试思路是根据组件所使用的技术，构思相应的测试想法。例如，测试人员根据图4-10所列举的技术，可获得如下测试想法。

- **Silverlight应用通过异步通信与服务端通信**。如果用户连续点击控件，客户端会向服务端发送多个间隔很短的请求，服务端如何处理这些接近“并发”的请求？如果服务端密集返回多个响应，客户端如何处理这些接近“并发”的响应？如果服务端返回的响应不符合请求的顺序，客户端如何处理？
- **服务端的网站和Web服务使用WCF技术**。WCF的默认配置对所传输数据的长度有较严格的限制。那么，被测系统使用了何种WCF配置？该配置对数据传输的限制是多少？能否构造一个大型报表，使其数据长度超过当前限制？
- **同步工具和Windows服务可能同时访问数据库**。当同步工具写入数据时，Windows服务会不会读到“脏数据”，以至于生成错误的报表？如果同步工具写入数据时会锁住表格，Windows服务会不会被阻塞，以至于实际用户体验到性能问题？如果Windows服务读取数据时会锁住表格，同步工具会不会被阻塞，以至于遇到数据库超时的错误？

以上所列仍旧是可行测试想法的一小部分。测试人员需要了解实现技术的特征和常见问题，才能全面思考，构思出有针对性的测试。

此外，测试专家Michael Bolton和James Bach针对部署图，提出了一批测试指导词，对于测试设计有很高的参考价值[Bolton09a]。其基本元素是节点、组件和路径。

节点是系统的组件，是测试设计的基本对象。

- **缺失和退出**：如果组件退出，系统会如何？
- **额外和干扰**：如果出现额外的组件或组件发出干扰，系统会如何？
- **错误**：组件会发生哪些错误？如果组件出错，系统会如何？
- **时间和顺序**：考虑时间、时序、时长、时限、速度、频率等时间因素对组件的影响。
- **内容与算法**：考虑组件容纳的数据和使用的算法。
- **条件行为**：在不同的条件下，考察组件的行为。
- **局限**：组件有什么业务和技术局限？这些局限对系统有何影响？
- **错误处理**：组件如何发现并处理自身的错误？如何发现并处理来自其他组件的错误？

线条连接了组件。测试人员需要针对组件之间的通信设计测试。

- **缺失和退出**：如果组件间的通信中断，系统会如何？
- **额外和分支**：能否建立额外的通信渠道？
- **错误**：组件间的通信会发生哪些错误？如果通信出错，系统会如何？
- **时间和顺序**：考虑时间、时序、时长、时限、速度、频率等时间因素对通信的影响。
- **状态通信**：组件之间如何传播或同步彼此的状态？
- **数据结构**：通信使用何种数据结构、数据格式、数据类型？

路径是线条构成的通路。测试人员需要针对图中的路径（通常对应为数据传播路径或工作流）设计测试。

- **最简的**：设计连接组件的最简单路径。
- **常用的**：考虑用户使用系统的常见路径。
- **关键的**：考虑完成业务的关键路径，例如完成某项业务有几条可行路径，关键路径不包含或只包含最少的非必经节点。
- **复杂的**：考虑复杂的数据流和工作流对系统的影响。
- **病态的**：考虑可能的病态路径，例如恶意数据在系统内传播的感染路径，又例如由于错误操作导致的异常工作流。
- **挑战的**：考虑对系统而言有挑战性的路径，例如伴随超高负荷运算的工作流。
- **错误处理**：系统能否发现并处理数据流和工作流中的错误？
- **周期的**：考虑周期的、定期的、间歇的事件对路径的影响。

在测试指导词的基础上，Michael Bolton利用部署图和测程来组织测试[Bolton12]。例如，项目时间允许执行40个测程。为了更好地安排测程，他在白板上画了如图4-11所示的部署图，并拟定了6个主要的测试策略：观察并建立测试先知、控制/变化/改变、强制失败、利用工具探索、探索数据或对象、探索活动。这些测试策略将覆盖产品的不同方面。

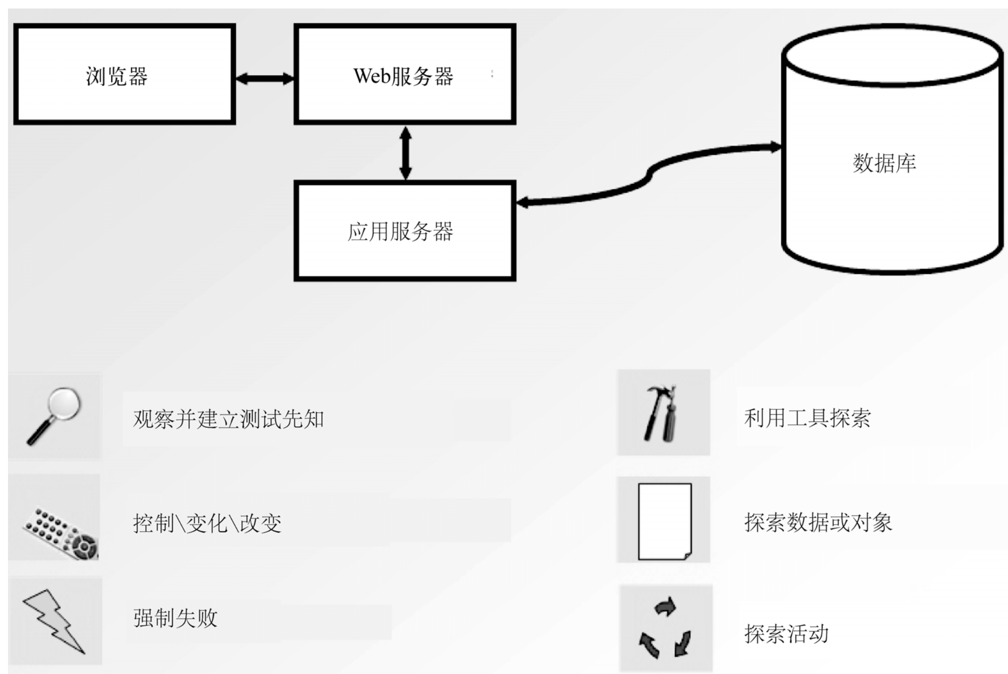


图4-11　部署图和测试策略

然后，他组织测试小组一起评审部署图。他们用即时贴代表测程，在一张即时贴上写下简单的主题词，然后贴到一个节点或线条上。因为即时贴是有限的，测试小组在张贴过程中，要平衡不同测试策略的数量，要平衡在不同节点和线条上的测程。当贴完40张即时贴时，测试安排就自然结束。如图4-12所示，测试小组获得了一个覆盖整个系统的测试方案。

在分配测程时，测试小组要考察部署图的完整性。任何模型都是对产品的简化，所以它会省略一些内容。如果部署图遗漏了某个节点或线条，那么测试安排就会产生漏洞。为此，测试人员要不停地自问：这幅图缺少了什么？现有测程和测试策略是否覆盖了整个系统？

由本节的讨论不难看出，系统生态图综合了黑盒分析和白盒分析。在黑盒层面，它列举了软件系统的用户和外部依赖，让测试人员思考系统的边界和接口。在白盒层面，它呈现了系统的主要组件和组件之间的联系，让测试人员可以分析整个系统的运作。因此，创建系统生态图有助于测试人员从整体上把握产品，并设计多样化的测试想法。

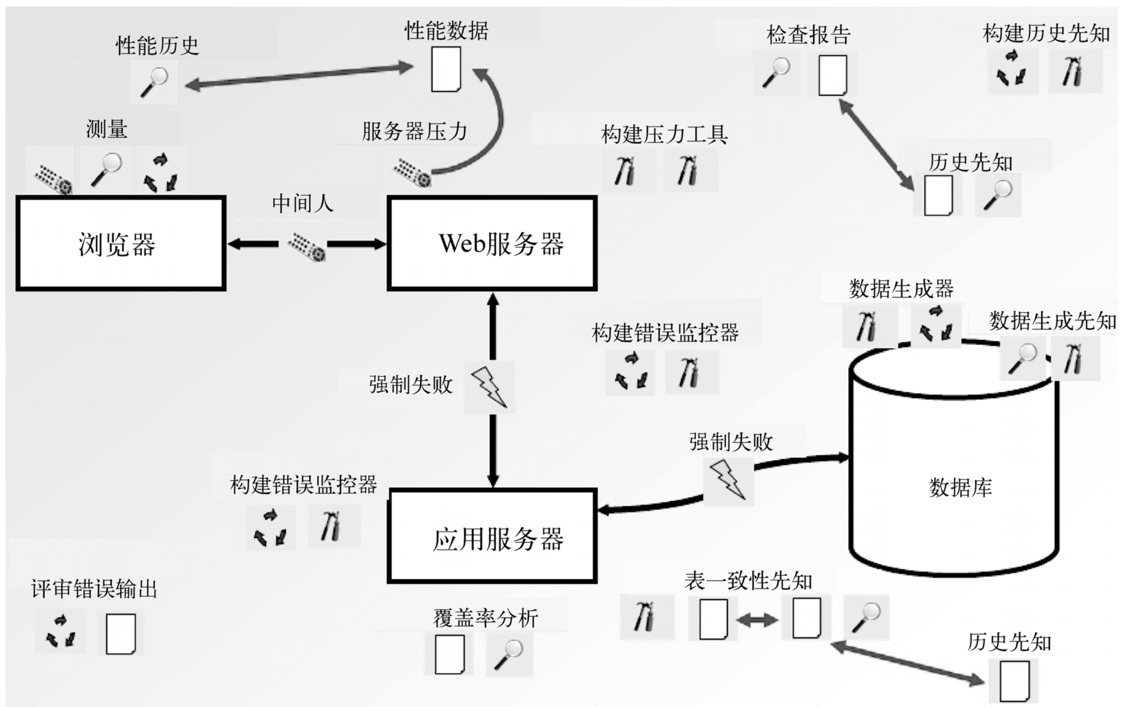


图4-12 用部署图安排测程

4.2.4 实体关系模型

实体关系模型（Entity-Relationship model，ER模型）是一种数据库建模方法，由计算机科学家陈品山（Peter Pin-Shan Chen）于1976年提出[WikipediaERM12]。ER模型被广泛地用于关系型数据库的分析与设计，并被推广到普通软件的数据分析。

图4-13是一个简化的在线交易系统的ER模型。其中，长方形节点是实体，代表系统中的一类数据，菱形节点是关系，反映了实体之间的关联。

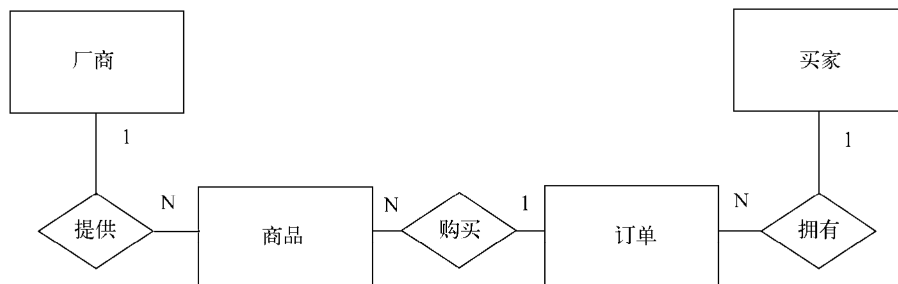


图4-13 在线交易系统的ER模型

- **通常，实体是系统中的“名词”，代表了业务领域的一类对象，这些对象拥有独立的含义和可被识别的标识**。例如，在线交易系统中，厂商、商品、订单、买家等实体拥有明确的含义、属性和标识，是构成交易的基本元素。实体的属性是实体的数据，例如“厂商”会拥有厂商编号、厂商名、联系人、地址、电话等属性。
- **关系往往体现为“动词”，联系了两个或多个实体**。例如，关系“提供”就反应了“厂商”对“商品”的供应关系。在关系“提供”的两条连线上标注了实体间的数据联系：“厂商是1，商品是N”，代表一位厂商可以提供多种商品，即厂商与商品是“一对多”关系。“厂商—提供—商品—购买—订单”，连接了厂商和订单。因为一位厂商可以向多份订单供货，一份订单可以包含来自多位厂商的商品，所以厂商与订单构成了“多对多”关系。关系也拥有属性，例如“购买”关系的属性包含商品的售出价格、税值等。

从黑盒测试的角度，测试人员可以使用ER模型来建立软件的数据模型。他分析业务文档，识别出业务对象，将它们抽象为实体。他还可以分配一个测程，其主题为“利用买家的用户界面，探索购物流程，以获得买家相关的ER模型”，即通过测试来创建ER模型。以下是一些识别实体和属性的技巧。

- **在Web应用中，一张表单往往对应了一个实体，表单的字段对应了实体的属性**。例如，用户注册表单对应了“用户”实体，需要填写的注册信息对应了用户属性。又例如，订单确认页面暗示了“订单”是一个实体，页面上的金额、下单时间、支付方式等信息是订单的属性。此外，订单页面包含指向商品的链接，暗示了“订单”与“商品”之间存在“购买”关系。
- **设置对话框常常呈现了一个实体拥有的属性**。例如，图4-3展示了PowerPoint对图片阴影的设置对话框，它表明图片是一个实体，并暗示阴影也是一个实体，而颜色、透明度、大小等具体设置是阴影的属性。
- **有些软件或服务用对象模型、Web服务等方式提供了编程接口**。其中，一个类型常常对应一个实体，类型的字段或特性对应了实体的属性，类型之间的聚合关系往往暗示了ER模型的关系。

在测程中，测试人员可以用简化的ER模型图来记录建模结果。例如，图4-14是一个简化的ER模型图，它用连线来表达关系，只对重要的关系标注了内容。

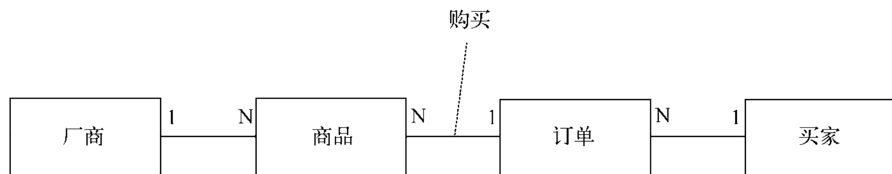


图4-14　简化的在线交易系统的ER模型

从白盒测试的角度，测试人员可以分析产品数据库（如图4-15所示）来建立ER模型。通常，

一张表对应一个实体，外键定义了实体之间的关系。Customer表对应“用户”实体，Order对应“订单”实体，由Order到Customer的外键对应了“拥有”关系。Purchase是一张关联表，它体现了Order与Product的“多对多”关系，反映了“购买”关系。



图4-15 在线交易系统的数据库设计

在获得ER模型之后，测试人员可以综合使用CRUD（Create，Read，Update，Delete）[Hendrickson13]、出租车漫游、快递漫游、沙发土豆漫游 [Whittaker09]等手段来实施测试。

CRUD就是对实体、关系和属性进行创建、读取、更新和删除操作，是所有数据测试的基本步骤。围绕这些操作，测试人员可以构思出多样的测试想法。以下是一些可能的策略。

- **沙发土豆漫游**。“沙发土豆”是一句美国俚语，比喻一个人很懒，整天陷在沙发里看电视。James Whittaker以此为隐喻，提出了沙发土豆漫游。运用该漫游测试时，测试人员总是不修改软件提供的默认值，只是不停地点击“下一步”让软件处理它给出的默认值。有时，软件的用户界面和后台计算是不同的程序员编写的，他们可能使用不同的逻辑去检查数值的合法性，因此用户界面上给出的默认值有可能是后台计算不能处理的非法值。此外，沙发土豆漫游还可以发现一些软件应该给出默认值（以减少用户的输入负担），但是没有给出默认值的情况。
- **测试人员还可以装成另一种“沙发土豆”——拒不提供数值**。例如，网页表单给出了一批字段的默认值，测试人员删除其中一些默认值，然后提交表单。面对这种情况，不同的实现有不同的做法：有些网页会检查字段的数值，并报告数值缺失的错误；有些后台服务器会检查用户提交的数值，并报告错误；有些后台服务器会使用它认为的默认值（也许与网页显示的默认值不同）。根据网页的实际表现，测试人员可以再设计新的测试。
- **对于实体和关系的属性，测试人员需要测试非法值和极限值**。例如，图4-3展示了PowerPoint对图片阴影的设置。测试人员需要针对每个控件（阴影的属性）测试非法值和极限值。
- **有时，实体的属性对于某些模块是合法的，对于另一些模块却会导致错误**。例如，在图4-3所示的对话框中，测试人员设置了一些阴影属性的极限值（如将“模糊”设为0磅）。此时，PowerPoint能正确绘制阴影。随后，测试人员将幻灯片另存为PDF文档，却发现所生成的文

档不能正确显示阴影。为了发现这种错误，测试人员需要考虑快递漫游。快递漫游的隐喻是快递人员会携带着货物穿越万水千山、城市乡村、大街小巷。相似地，漫游测试要求测试人员随着数据游历整个系统，跟着实体进入系统的不同部分，并在各处调用CRUD。

- **软件可能提供了多种方式来修改实体、关系和属性**。例如，用户可以用如图4-3所示的PowerPoint对话框来修改图片的阴影效果，还可以编写宏，用代码来修改图片阴影。为了保证测试覆盖，测试人员需要考虑出租车漫游。出租车漫游的隐喻是出租车司机对城市交通非常了解，他可以想出很多路线到达目的地，而漫游测试需要覆盖所有这些路线。对于复杂的软件，测试人员需要多方面调查功能的接口，穷尽所有的接口来调用CRUD。

除了以上测试策略，测试人员还要是考察CRUD对ER模型的影响。最常用的启发式方法是通过CRUD去制造“零个、一个、多个”实体的情况 [Hendrickson13]。以下是一些常见的启发式问题。

- **能否创建一个没有依赖的实体**？在如图4-13所示的在线交易系统中，通过注册新买家可以创建一个没有订单的买家。那么，可以创建一个没有买家的订单吗？可以先创建一个买家，让他下一笔订单，再删除该买家，从而获得没有用户的订单吗？合理的数据库设计可以禁止没有用户的订单，不过测试人员不能假定数据库设计正确。此外，软件的一些模块也许会短暂地使用这样的订单，这种“非法”且“临时”状态的订单会遗漏到其他模块并导致系统出错吗？测试人员可以使用出租车漫游和快递漫游去调查。
- **能否删除一个有依赖的实体**？在图4-13的ER模型中，如果删除一个有订单的买家会发生什么？系统是禁止删除有订单的买家，还是会级联删除该买家拥有的订单？如果订单已经付款且没有交货，级联删除订单会不会导致无法交货？如果一家厂商退出市场，但是它的一批货物仍旧保存在电子商城的配送中心（电子商城已经为这批货付款），那么该如何处理？能否删除该厂商？如果删除厂商会导致相关产品被级联删除，那么电子商城还能否继续销售这篇存货？这些问题看似极端，但是现实世界的电子交易比图4-13的ER模型要复杂得多，会遇到更多的难题。测试人员需要从买家、厂商、电子商城等不同用户的角度考虑可能的业务流程。
- **能否改变实体之间的关系**？例如，电子商城通过快递公司发货给买家A。货在途中时，买家A向电子商城报告，这笔交易是有人盗用其账户所为，要求撤单。此时，电子商城能否要求快递公司停止向买家A送货，转而将该货品送给另一位购买相同商品的买家B？如果这在业务上是可行的，那么软件系统如何支持这笔修改？如何修改商品、订单、买家之间的联系？

综上所述，ER模型用简练的图形描述了软件系统的数据关系，帮助测试人员概览业务模型（黑盒视角）或实现模型（白盒视角）。利用该模型和相关测试方法，测试人员可以构造出有力的操作和流程去挑战软件系统。

4.2.5 状态机模型

状态机（state machine）是一种常见的软件建模方法，被广泛地应用于软件分析、设计、开发与测试，并且是基于模型的测试[1]的重要技术之一[WikipediaMBT12]。

3.2.7节介绍了一篇Windows 8应用的测试指南，该文档以应用的状态为核心，介绍了相关知识和测试要点。图4-16是从操作系统的角度考察应用状态所获得的状态图 [Hilo12]，其图元的含义如下。

- 节点代表Windows 8应用的状态，包括从未运行、不在运行、被用户关闭、运行、已挂起和已终止。
- 线条代表变迁，其起点是变迁的源状态，终点是目标状态，线条上的文字是变迁的触发事件。例如，当用户从当前应用切换到开始屏幕时，操作系统向处于“运行”状态的应用触发“挂起”事件，并在5秒后将其置于“已挂起”状态。

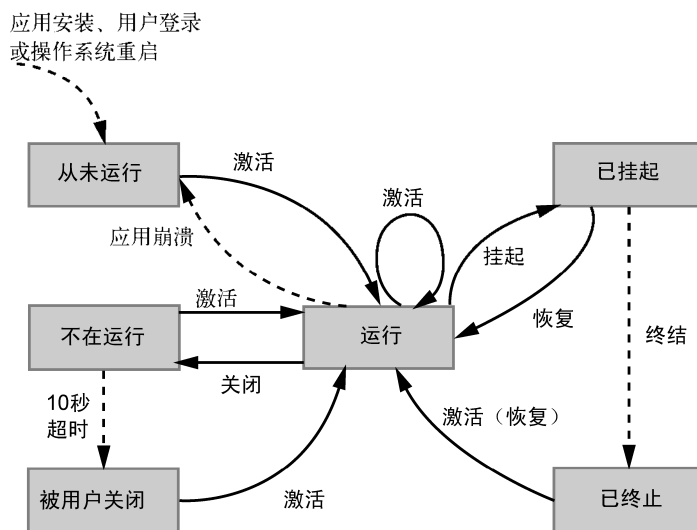


图4-16 Windows 8 应用的状态图

在获得状态图之后，测试人员可以从以下角度设计测试用例。

---

① 该“基于模型的测试”是指测试人员建立软件的形式化模型，通常是有限状态自动机、马尔可夫链、形式化系统等数学模型，然后利用可执行路径搜索、定理证明、符号执行、模型检查等算法，自动地执行测试或检查软件模型。这是一类基于形式化模型的自动化测试技术，是“狭义”的基于模型的测试。除本句之外，本章讨论的是“广义”的基于模型的测试，即利用软件的模型（可以是形式化模型，也可以是非形式化模型），进行测试设计。本章的讨论偏向人工设计测试想法，但并不排除使用自动化算法来实施测试。我建议读者在与其他人讨论“基于模型的测试”时，要了解对方所说的是“狭义”概念还是“广义”概念，这有助于掌握语境，避免误解。

- **设计测试用例以覆盖所有状态**。对于图4-16，测试人员应该操作被测应用，使它抵达所有状态节点：从未运行、不在运行、被用户关闭、运行、已挂起、已终止。
- **设计测试用例以覆盖所有状态变迁**。对于图4-16，测试人员应该操作被测应用，使它经过所有变迁线条。例如，当应用处于“运行”状态时，测试人员切换到Windows桌面，然后在任务管理器中检查它正确地进入到“已挂起”状态，这覆盖了“挂起”变迁（“运行”状态→“已挂起”状态）。
- **设计测试用例以覆盖所有触发事件**。有些变迁可以被多个触发事件激发，测试人员应该测试所有的触发事件。例如，为了覆盖“挂起”变迁（“运行”状态 →“已挂起”状态）的触发事件，测试人员可以测试：从被测应用切换到Windows桌面、从被测应用切换到Windows开始屏幕、从被测应用切换到其他Windows 8应用、合上笔记本电脑（或按下Windows平板上的开机键）令计算机休眠等。

在许多软件中，对象的状态是显而易见的。例如，经常在线购物的用户都了解订单具备尚未提交、已提交未确认、已确认、发货、交货等状态。需求文档或设计文档常常描述了重要对象的状态，甚至绘制了状态图。测试人员可以直接使用这些信息来构建状态机，以减少建模的开销。不过，即便项目经理或程序员提供了状态机，测试人员仍需要通过测试去发现隐藏的状态和相应的变迁。这是因为软件是高度复杂的，任何状态机都只是一种简化表达，它可能遗漏了某些值得测试的状态或变迁。为此，测试人员需要在测试迭代中检验并完善状态机。

状态机建模的第一步是识别状态。如果将软件状态看作内部变量拥有的值，那么软件就有无穷多的状态。注重实效的测试人员会根据测试使命明确建模目标，从而将无穷多的值归纳到少数几个状态。例如，图4-16的重点是分析Windows 8应用与操作系统的交互，所以该模型将应用正常运作的情形都归纳为“运行”状态，并以它为中心分析应用的其他状态与变迁。如果测试重点是应用的业务逻辑，那么测试人员可以忽略图4-16的所有状态和变迁，直接深入到“运行”状态中去建模。

在测试过程中，测试人员可以参考以下技巧[Hendrickson13]，来识别软件状态。

- **如果软件展现了新的界面，允许执行更多的命令，那么它进入了另一个状态**。如图4-17所示，在PowerPoint中选中一个图片，Ribbon界面会显示新的标签（图片工具 — 格式），允许执行图片相关的命令。这暗示软件进入了“图片编辑”状态。



图4-17　选中图片后，Ribbon显示新标签

- **如果软件展现了新的界面，可执行的命令变少，那么它进入了另一个状态**。在如图4-17所示的情况下，点击幻灯片中的文字，“图片工具”标签会消失。这暗示软件离开了“图片编辑”状态，进入了“文字编辑”状态。另一个例子是，在订单提交之前，用户可以修改订单中的商品，一旦订单提交，用户就无法修改订单内容。这暗示订单从“未提交”状态进入“已提交”状态，触发事件是“提交”。
- **如果相同的操作导致了不同的结果，那么软件进入了另一个状态**。例如，向一个固定容量为5个单元的空队列连续添加数据。前5次添加操作都会成功，第6次操作会导致错误。这说明“容量耗尽”是一个独立的状态。同理，“没有数据”也是一个独立的状态，因为向空队列索取数据会导致错误。
- **分析需求文档和用户界面，寻找“当……的时候”、“在……之后”、“在……之前”的表述，这常常暗示软件处于特定的状态**。例如，“在订单提交后，用户不能再增加或删除商品”就暗示“已提交”是一个状态。又例如，Windows安装补丁时会提示“正在更新您的系统，请不要关机”。这暗示“正在安装更新”一个重要的状态。
- **如果软件显示等待提示符或进度条，那么软件正在进行某项任务，这段任务时间可以看作一个状态**。例如，启动Windows 8的Bing新闻应用，该应用会显示几秒钟的启动界面（如图4-18所示）。这暗示在图4-16的“不在运行”和“运行”之间隐藏了一个状态：“正在初始化”。发现该状态后，测试人员会考虑：能不能中断“正在初始化”状态？如果中断该状态，会不会导致数据丢失？



图4-18 Bing新闻的启动界面

除了识别状态，测试人员还需要发掘隐藏的变迁，有时新发现的变迁会揭示出新的状态。以下是一些识别变迁的基本观察点[Hendrickson13]。

- **用户命令会触发变迁**。例如，用户在PowerPoint中点击图片，能够触发变迁，使软件进入“图片编辑”状态。

- **软件自身会触发变迁**。例如，Bing新闻应用在启动时会进入“正在初始化”状态。如果不被打断，Bing新闻应用会在几秒内结束该状态，进入“运行”状态。
- **软件的外部环境会触发变迁**。在图4-16中，当应用处于“已挂起”状态时，操作系统可能触发“终止”事件，使应用进入“已终止”状态。另一个例子是，用户使用智能手机的GPS导航软件来辅助驾驶。当他驾车通过隧道时，GPS信号可能中断，于是导航软件会进入一个特殊的状态。当他驶出隧道时，GPS信号恢复，导航软件应该回到正常工作状态。测试人员应该检查这样连续的变迁所产生的影响。
- **时间会触发变迁**。例如，某软件查询数据库，查询操作的超时是30秒。如果该查询的执行时间超过30秒，那么查询所调用的数据库API会抛出异常，导致查询中断。从状态机的角度，这是超时事件导致软件状态变迁。再例如，许多软件会使用定时器，周期性（如每周、每天、每小时）地启动计算任务。开始计算、结束计算和计算产生的后果都可能触发变迁。

由以上讨论不难看出，构建完整的自动机模型并非轻而易举。测试人员需要探索软件来发现状态和变迁，并利用新发现去发掘更多的状态和变迁。这样的探索与发现之旅本身就是很好的测试。当测试人员从各个角度尝试触发变迁，软件会经历多项考验，一些隐藏的错误就会暴露出来。在整个建模过程中，测试人员可以考虑如下策略来加速探索和测试。

- 使用出租车漫游和出租车禁区漫游[Whittaker09]来发现更多的状态和变迁。出租车漫游要求测试人员用不同的路线来达到同一个目标。利用该隐喻，测试人员需要发现到达（或离开）某个状态的所有变迁及触发事件。出租车禁区漫游要求测试人员尝试多种路线以前往不允许的区域。利用该隐喻，测试人员应该瞄准一个很难达到的状态，通过各种手段去构造一条或多条状态变迁路径，以到达该状态。
- 因为状态图是一种联通图，状态和变迁构成了一组路径，所以测试人员可以参考Michael Bolton针对路径提出的测试指导词（参见4.2.3节），通过构造最简的、常用的、关键的、复杂的、病态的、挑战的路径来测试系统。在遍历路径时，要考察错误处理、故障恢复、周期性事件、间隙性事件等因素对状态机的影响。对于复杂的状态图，可能的路径是海量的。在设计路径时，要基于领域知识和产品风险，来选择有潜力发现错误的路径。
- 程序员在编写代码时，常常会忽略一些可能发生的事件，从而导致软件不能应对复杂的应用情景。为此，测试人员需要针对每一个状态，考虑打断该状态会不会导致故障[Hendrickson13]。例如，当用户离开一个Windows 8应用时，操作系统会给该应用5秒的时间去执行清理工作（通常是保存用户数据和程序状态）。5秒之后，无论清理工作是否结束，操作系统都会挂起该应用。如果此时清理工作还没有完成，用户数据可能会丢失。为此，测试人员需要对每个业务状态触发“挂起”事件，来测试应用能否在5秒内完成清理工作。再例如，一款智能手机软件提供了录音功能。当用户正在用它录音时，一通电话拨入，智能手机自动切换到通话界面。此时录音状态被中断，电话之前的录音可以被正确保存吗？电话结束后，软件能否自动恢复到录音状态，继续录音？

- 当测试人员获得一个相对完整的状态图后，他可以将其转化为状态表[Hendrickson13]。例如，表4-6是图4-16对应的状态表，第一列是源状态，第一行是触发事件，表的内容是源状态在触发事件作用下变迁到的目标状态。表中有一些空白单元格，说明状态图没有阐述相应的源状态和触发事件的行为。测试人员应该检查这些模型表述上的空白的实际影响。在源状态下触发事件会发生吗？这对源状态和触发事件会导致软件故障吗？

表4-6　用状态表来表示状态机

触发事件 / 状态	激　活	挂　起	继　续	终　止	关　闭	崩　溃	超　时
从未运行	运行						
不在运行	运行						被用户关闭
被用户关闭	运行						
运行	运行	已挂起			不在运行	尚未运行	
已挂起			运行	已终止			
已终止	运行		运行				

简而言之，构建和运用状态机模型可以帮助测试人员更好地理解软件的运行机制，更好地把握软件与外界的交互。这不但有助于探究软件的细节，还可以更全面地测试软件在不同情景下的行为。

4.2.6　多种多样的模型

在软件测试中，可选的测试模型是多种多样的。本书第3章就介绍了几种有价值的测试模型。

- Google ACC（3.2.2节）针对产品的属性、部件和能力建立测试模型，让测试设计聚焦于对用户最有价值的情景。此外，利用能力矩阵的热点图，测试人员可以快速识别产品的高风险区域，以针对风险投入测试资源。
- 功能列表（3.2.4节）是一种常见的功能测试模型。它用层次结构列举了产品的功能，既抽象出主要功能区域，又提供了必要的细节。利用功能列表，测试人员可以逐步发展测试设计，并周密地覆盖产品的功能。
- Fit表格（3.2.6节）用具体的测试用例诠释了业务规则。它让业务分析师的业务知识以自动化测试的形式传递给测试人员和程序员，从而使开发流程更加顺畅。
- 质量列表（3.2.9节）提示测试人员从多个方面考虑产品的质量，分析出值得重点测试的质量特性。

软件产品是高度复杂的，任何模型只能描述它的一个局部。因此，测试人员需要多种模型，

从不同角度考察软件。测试专家Rikard Edgren认为可视化地表达头脑中的模型能够帮助测试设计，切换多个模型能够产生差异化的测试想法[Edgren12]。

如图4-19所示，Edgren的测试对象是他开发的示例软件“完美年龄计算器”，当用户输入生日字符串，它会输出生日所对应的年龄。Edgren指出该用户界面是测试人员可利用的第一个模型。针对该模型，测试人员可以快速产生一批测试想法。

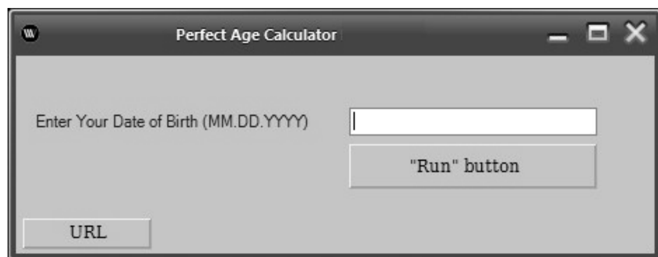


图4-19　模型一：“完美年龄计算器”的用户界面

- **测试窗体**：窗体可以被正确地缩放、最小化、最大化、关闭吗?
- **测试输入框**：输入框可以接纳多少字符？输入框如何处理错误的输入?
- **测试“运行”按钮**：点击该按钮，软件可以给出正确的计算结果吗？如果输入框为空，点击该按钮会发生什么?
- **测试“URL”按钮**：点击该按钮，软件可以打开正确的URL地址吗?

随后，Edgren给出了第二个测试模型。如图4-20所示，该模型对时间建模，将时间线划分成很久以前、现代时间、现在和未来。利用该模型，测试人员可以构造四类日期输入，以测试软件对年龄的计算。



图4-20　模型二：时间线

Edgren的第三个模型是对时间线模型的细化，其结果如图4-21所示。这是一个很有价值的模型，可以复用于其他时间相关的测试。

- 该模型提示测试人员考虑计算机可以表达的“时间的开始点”和“时间的结束点”。对于.NET程序，.NET内建类型System.DateTime的时间范围是[“1/1/0001 12:00:00 AM”，“12/31/9999 11:59:59 PM”]。测试人员可以测试这两个边界值，并测试早于“开始时间”的时间点和晚于“结束时间”的时间点。如果程序员设计了新的时间类型，测试人员可以针对该类型的边界进行测试。
- 该模型提示“闰年”和“闰日”是特殊时间点，值得测试人员考虑。
- 该模型提示今年、本月、今日是特殊时间点，值得测试人员考虑。

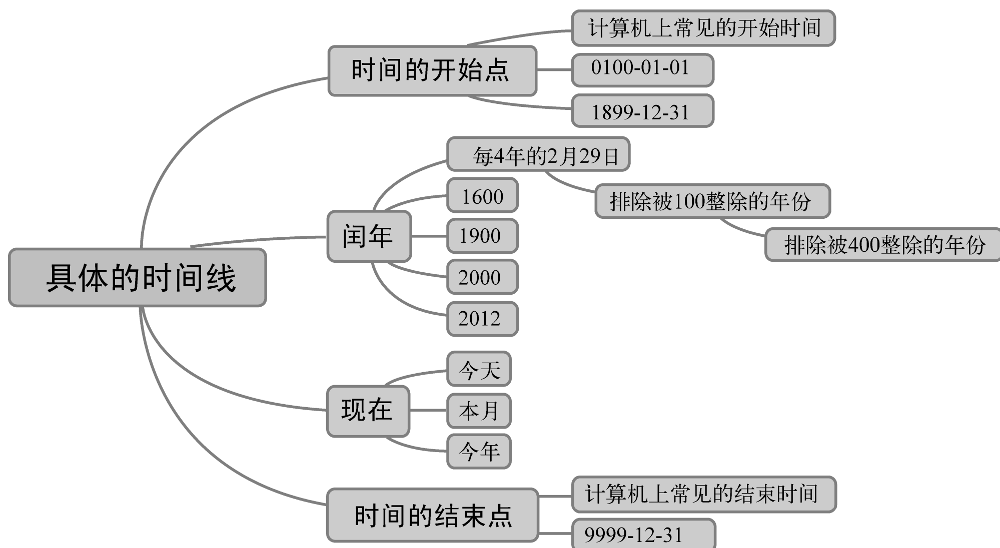


图4-21　模型三：详细的时间线

第四个模型针对日期字符串，将其划分为年、月、日三部分，分别设计测试想法。详细模型请参考图4-22。

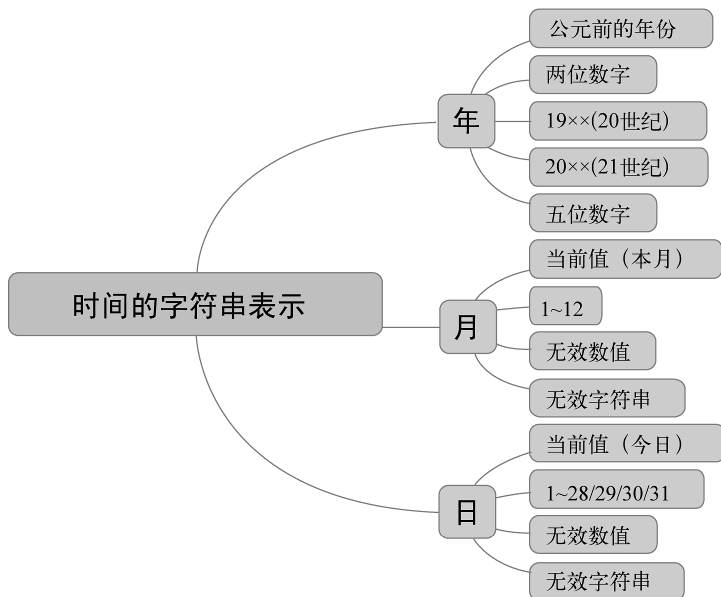


图4-22　模型四："年、月、日"的字符串表示

- **"年"的字符串表达：**公元前的年份、两位数字、19××、20××、五位数字。

- **“月”的字符串表达**：本月、1~12、无效数字、无效字符串。
- **“日”的字符串表达**：今日、1~28/29/30/31、无效数字、无效字符串。

随后，Edgren从实现的角度分析了软件，给出了如图4-23所示的技术流模型。利用该模型，测试人员可以从实现的角度设计测试。

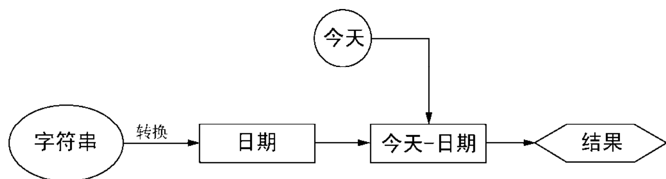


图4-23　模型五：技术流

- **字符串到日期的转化**：软件可以处理无效的字符串吗？软件可以处理本地化的日期字符串吗（如中文日期字符串“二零一三年一月一日”）？软件可以处理全角和半角字符吗？
- **获得当前日期**：软件如何获得当前日期？软件使用本地时间，还是UTC时间？
- **时差计算（今天 – 日期）**：软件用何种数据类型记录时差？该计算会导致整数溢出吗？
- **结果报告**：软件如何将时差数据转化成年龄字符串？软件如何报告用户输入的错误？软件如何报告计算过程中发生的错误？

最后，Edgren用思维导图对质量特性进行建模，具体模型请参考图4-24。该模型剪裁自The Test Eye发布的软件质量特性集[TheTestEye11]（参见3.2.9节），将测试思考聚焦于“完美年龄计算器”适用的质量特性上。

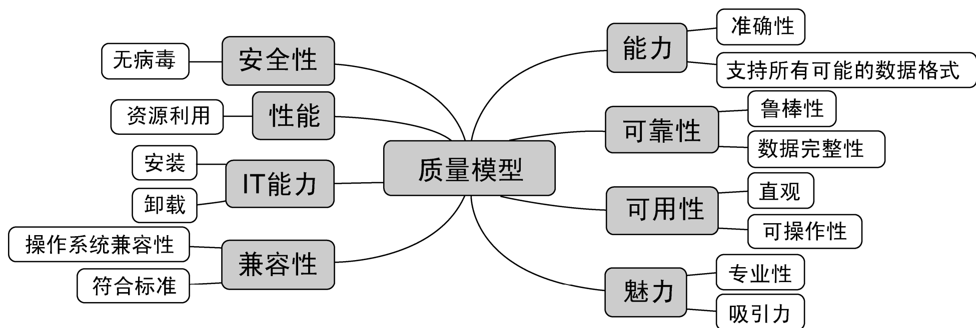


图4-24　模型六：质量特性

在以上讨论中，Edgren针对一款简单软件，创建了6个模型。面对复杂的软件，测试人员会创建更多的模型。许多测试模型只是存在于测试人员的头脑中，并没有记录下来。有时，这是合理做法，因为记录模型需要花费时间，直接使用简单的模型可以更快速地测试。有时，可视化头脑中的模型是更好的策略，它可以从以下几个方面促进更好的测试。

- **记录模型可以帮助模型的演化，简单的模型在测试过程中会发展成丰富的模型**。例如，图4-20的时间线模型发展出图4-21的详细时间线模型和图4-22的年月日模型。
- **一些测试模型可以应用于多个测试对象和产品，记录并复用这些模型能够提高测试效率**。例如与时间有关的测试设计都可以参考图4-21的详细时间线模型。
- **将模型记录下来，可以将它分享给测试小组**。这不但有利于知识和技能的传播，还可以收集同事对模型的反馈，以加速模型的演化。
- **用纸笔、思维导图软件等工具，测试人员可以灵活地记录模型**。只要付出很短的时间，就能收获有价值的模型。

统计学家George Box曾说过："本质上，所有模型都是错的，但是有些是有用的。"[WikipediaGEPB12]。这句被广泛引用的名言道出了建模的真谛。从测试建模的角度，这句话有3个方面的意义。

第一，所有的模型都是对软件的简化，是片面的、不完整的。如果测试人员只使用单一的模型或少数几种模型，他会遭遇认知偏差，从而错过重要的测试策略，并导致缺陷遗漏。测试人员需要用多个模型去分析软件，从而产生周密的、多样化的测试想法。

第二，正因为模型总是不完整的，测试人员不必追求"完美的模型"。他应该更注重模型的实用性，更多地考虑模型能否提供好的测试想法。

第三，在注重实用性的前提下，测试人员需要发现并消除模型的错误。在团队中分享模型并收集反馈意见是一个很好的方法。此外，他也应该在测试迭代中持续打磨模型——用模型指导测试过程，同时让测试过程来优化模型。

4.3　小结

本章介绍了测试建模的重要性，并介绍了一些常用的测试模型。

- 无论测试人员是否有意识地建模，任何测试都基于模型。然而，认真去构建好的模型，将提高测试过程的质量。
- 测试建模的基本任务是建立被测对象的模型，以帮助理解软件和设计测试。
- 纯粹的"通用模型"远离业务领域和设计实现，需要做"因地制宜"的改良，才能发挥作用。改良的基本方法是使用业务需求、产品元素、项目环境的信息去丰富和调整模型。
- 构建模型需要简化和聚焦，使用模型需要扩展和发散。
- 好的测试模型必须经过测试迭代的淬炼。
- "本质上，所有模型都是错的，但是有些是有用的。"
- 不要追求完美的模型，要创建注重实效的模型。模型是测试设计的工具，好的模型将有力支持测试设计。

- 不要依赖单一的模型，要综合多个模型，以避免认知偏差。不要完全信赖任何模型，要通过测试去质疑并改进模型。
- 启发式测试策略模型HTSM是一个指导测试设计和风险分析的概念框架，测试人员需要根据项目语境对其进行裁剪和丰富。
- 在测试中，功能列表（针对软件能力）、输入与输出模型（针对软件与外界的交互）、系统生态图（针对软件结构）、实体关系模型（针对数据关系）、状态机模型（针对软件状态）、质量特性列表（针对多样化的质量）等是常见的测试模型。
- 测试人员可以针对特定领域或测试对象开发出多种多样的模型。
- 条件分析是一种启发测试思路的常见方法。

第5章 测试技术

如果说测试是为了发现错误而执行程序的过程，那么测试技术就是运行程序并判断其对错的具体方法。高效的测试要求测试人员根据软件产品和项目环境选择合适的测试技术，并灵活机动地加以运用。假如测试人员只掌握少数几种测试技术，那么他很难作出合理的选择，也很难用多样化的测试来调查复杂的软件。为了高质量地完成测试任务，测试人员需要在工作中持续学习测试技术，分析它们的长处、不足和适用情况，以逐渐完善自己的测试技术体系。

本章首先介绍一个测试技术分类系统，以概览各类测试技术。然后，面向测试设计和执行讨论一批有价值的测试技术。

5.1 测试技术分类系统

本节介绍Cem Kaner教授建立的测试技术分类系统[Kaner01][Kaner11]。该分类系统经过十余年的发展，比较成熟，对测试技术的理解和选择很有帮助。

Cem Kaner认为测试技术会从7个方面对测试过程进行指导。

- **范围**：测试的对象。例如，功能测试的测试对象通常是一个具体的功能或特性。
- **覆盖**：测试的程度。例如，如果测试时间很有限，测试人员通常只能测试一个功能的主要场景；如果测试时间充裕，他会测试更多的场景和操作方式，以扩大测试覆盖。通常，测试人员会统一考虑测试的范围和覆盖。
- **测试者**：由谁来执行测试。例如，许多软件厂商提供Beta版本供用户试用，以便收集他们对早期版本的意见。对于Beta测试，其测试执行者就是软件试用者。
- **风险**：测试要去发现的潜在问题。例如，Google ACC所重点侦测的风险是产品的功能不能提供有竞争力的特性。
- **活动**：测试如何执行。例如，两因素组合测试规定测试用例集要覆盖任意两个因素的取值组合。
- **评估和测试先知**：如何评价测试是否通过。例如，测试人员会使用软件的先前版本来判断软件的行为是否向后兼容。

- **结果导向**：测试的目标。例如，BVT（build verification testing）的目标是检查构建是否足够稳定可用于更大范围的测试。

任何测试活动都会涉及以上所有方面，而一个具体的测试技术通常只着力于其中的1~3项。例如，组合测试算法只讨论如何生成测试输入数据，即只覆盖了“活动”的一部分，对于其他方面没有讨论。为了应用测试技术，测试人员不但要知晓它解决了哪些问题，更要了解它没有解决哪些问题。在实际测试中，测试人员需要根据项目语境，补上相应的空白。

利用以上测试驱动因素，Cem Kaner建立了一个六要素测试分类系统。因为许多技术考虑了多个驱动因素，所以它们会属于多个分类。这提示测试人员要多角度地考虑测试技术，选择适合当前任务的切入点去应用它们。以下是该测试分类系统的六大分支。

第一，基于覆盖的测试技术关注测试的范围和覆盖。典型的测试技术如下。

- **功能测试**：覆盖产品的功能。
- **功能或特性的集成测试**：覆盖功能之间的交互。
- **漫游测试**：其基本策略是搜索产品的一个区域，并收集相关信息（5.4节将介绍漫游测试）。这通常要求覆盖指定的一组测试对象。
- **等价类分析**：覆盖所有的等价类。
- **边界测试**：覆盖边界值和邻近区域。
- **最佳代表测试**：要求从众多测试用例中选择最有代表性的测试用例，常见的例子包括用户最可能使用的数据、给产品最大挑战的数据、最能暴露风险的数据等。
- **域测试**：从一个数值集合中选择测试输入数据，等价类分析和边界测试是典型的域测试技术。如果穷尽测试不可行，那么域测试要选择出典型的测试数据，然后分别加以覆盖。
- **测试想法目录**：包括测试想法列表、质量特性列表、缺陷目录等启发式想法集合。测试人员要依次检查每个想法，选择合适的想法来设计具体的测试。
- **逻辑表达式**：覆盖逻辑表达式的所有可能输入。
- **多变量测试**：使用某种覆盖标准去测试多个变量的取值组合，组合测试是一种多变量测试技术（参见4.1.1节）。
- **状态变迁测试**：覆盖状态机中所有的变迁（参见4.2.5节）。
- **用户界面测试**：覆盖所有用户可访问的界面元素。
- **基于规格说明的测试**：检查规格说明的所有陈诉得到正确地实现。
- **基于需求的测试**：检查所有需求被正确地实现。
- **依从性测试**：检查软件符合所有需要满足的法律和约定。
- **配置测试**：覆盖所有或典型的软硬件配置组合。
- **本地化测试**：覆盖所有被本地化的元素，包括界面字符串、日期格式、货币格式、字符集等。

第二，基于测试者的测试技术关注谁来执行测试。

- **用户测试**：让实际用户测试产品。通常，测试人员与用户一起工作，收集用户发现的缺陷和提出的意见。
- **Alpha测试**：让产品的早期用户（通常是公司同事）试用产品。
- **Beta测试**：让一批实际用户测试产品。测试人员不与用户共同测试，但会分析用户提交的缺陷。
- **缺陷大扫除**：邀请项目团队的所有人员参与测试，旨在让不同的人通过不同的视角来检查软件，从而在短时间内（缺陷大扫除通常持续1~2天）发现大量的缺陷。
- **专家测试**：邀请领域专家和测试人员结对测试。
- **结对测试**：让测试人员和测试人员结对、测试人员和程序员结对，从而产生更多差异化的测试想法。
- **内部试用**：让项目团队在日常工作中使用自己开发的产品，从而尽早发现实际用户会遇到的问题。
- **本地化测试**：让熟悉目标市场文化的人来测试软件。例如，Microsoft Word团队的成员来自五湖四海。团队领导通常会安排来自中国的员工执行Word中文版和中文输入法的专项测试。

第三，基于风险的测试技术关注潜在问题。

- **边界测试**：其关注的是软件在处理边界值时很可能出错的风险。
- **快速测试**：一组很轻便的测试方法，针对一组典型的软件错误实施攻击（5.5节将介绍快速测试）。
- **约束测试**：利用输入约束、输出约束、计算约束和数据约束对软件进行测试。这些约束往往涉及软件能力的边界，会暴露软件的不足（5.3.3节将介绍约束检查）。
- **逻辑表达式**：关注多个条件变量的组合，以检查软件能否处理一些罕见的情况。
- **压力测试**：关注软件如何应对远远超过正常工作量的负荷，此时一些隐藏的缺陷很可能会暴露。
- **负载测试**：关注软件能否有效地使用资源去处理工作负载。其针对的风险是软件不能合理地使用资源，当工作负载还没有到达预期上限时，它已经占用了太多的资源，且处理速度变得很慢。
- **性能测试**：关注软件能否在可接受的时间内完成计算任务。其针对的风险是软件的反应速度不能满足用户的期望。
- **基于历史的测试**：要求测试人员分析先前版本中所发现的错误，识别出可能再次复现的缺陷，利用它们去设计测试。其针对的风险是程序员会犯同样的错误，或者某些缺陷描述了业务或技术面临的根本性困难，很难彻底解决。
- **基于风险的多变量测试**：从风险的角度测试多个变量的取值组合。例如，4.1.2节在测试

Word的高级对话框时，建议测试默认设置和只更改了一个配置项的设置，因为它们更贴近用户的使用情景，更容易暴露真实用户会遇到的问题。

- **可用性测试**：检查软件是否容易学习和使用。其关注的风险是糟糕的软件设计让用户感到挫折，以致他们会使用竞争对手的产品。
- **配置和兼容性测试**：检查软件在不同软硬件平台上的表现，所针对的风险是产品可能在某些平台上失败。
- **互操作性测试**：检查软件与相关系统的交互。因为几个系统通常由不同的团队开发，他们对交互协议可能有不同的解读，所以系统交互总是会暴露出各种复杂的问题。
- **长序列测试**：反复且随机地运行一组测试用例。因为测试序列漫长且覆盖面广，它可能暴露内存泄漏、竞态条件、悬挂指针等问题。

第四，基于活动的测试技术关注如何执行测试。

- **游击测试**：测试人员在固定的时间盒内，对软件的特定区域实施基于风险的测试。
- **两因素组合测试**：测试人员需要生成符合两因素组合覆盖标准的测试数据。通常测试人员会利用组合测试工具生成测试数据。
- **随机测试**：利用随机数生成器去产生测试数据、安排测试顺序或选择测试对象。
- **用例测试**：根据用例或序列图来设计测试序列，以覆盖软件的操作序列。
- **情景测试**：创建一个或一组故事来测试软件在故事场景中的表现。
- **安装测试**：在不同的平台上安装和卸载软件，并检查操作结果。
- **回归测试**：运行已有的测试用例来再次测试相同的功能。
- **长序列测试**：要求测试整晚运行或持续几天，以发现一些短时间测试不能发现的问题。
- **猴子测试**：要求测试人员构建自动化测试程序，来随机地、持续地、自动地测试产品。例如，测试程序基于产品的自动机模型和当前状态，随机地选择触发事件，以触发状态变迁，并持续迭代下去。在此过程中，测试程序会判断变迁是否正确、产品是否出现错误。
- **性能测试**：要求测试人员识别用户使用产品的典型模式，确定需要评估的性能指标，然后构造出相应的测试流程去考察软件的性能。

第五，基于评估的测试技术关注如何判断测试通过。

- **功能等价测试**：比较被测软件的行为和参考软件（例如已经发布的上一版软件）的行为。如果被测软件的行为与参考软件不一致，那么它很可能出现错误。
- **数学先知**：根据数学规则判断测试结果的对错。例如，三角函数、求平方根、矩阵转置等计算的结果都需要满足数学定理和计算法则。
- **约束检查**：根据一组规则判断测试是否通过。例如，测试随机数生成函数时，测试人员很难判断每一个输出是否正确。他可以持续调用该函数，收集一大批随机数结果，然后分析这批数据。如果数据分布足够均匀，不存在明显的规律，那么该函数的质量就较好；

如果数据分布呈现出某种规律，以致测试人员可以预测下一次输出的取值，那么该函数还需要改进。

- **自检验数据**：包含可以用于检验的数据。例如，为了暴露对数据的异常修改，测试人员计算了数据的哈希值，让它随数据在系统中传播。每经过一个模块后，测试程序会重新计算数据的哈希值，并与先前保存的值进行比较。如果两个值不同，那么数据被该模块篡改。
- **比较已保存的结果**：通过对比前后两个版本产生的数据，来侦测新版本可能引入的错误。
- **比较规格说明或其他权威文档**：根据权威文档来判断软件的行为是否正确。
- **基于诊断的测试**：会使用软件的调试功能或一些调试辅助工具。当调试功能被打开或辅助工具被启用时，这些面向诊断的代码会持续检查产品的状态。当测试人员运行测试时，它们可以发现产品的异常状态，如内存泄漏、句柄泄漏、未处理系统调用错误等，并报告给测试人员。
- **可检验的状态模型**：使用测试人员构建的状态机为测试先知。测试程序调用软件接口，以驱动状态变迁，然后比较软件的实际状态和预期状态，以检查可能的错误。

第六，结果导向的测试技术关注于特定目标或文档。

- **构建检验**：其目标是判断当前构建是否足够好，能否提供给测试小组或更广泛的用户去测试。
- **确认测试**：运行精心设计的确认测试用例，其目标是证明软件满足预先定义的要求。
- **用户接受测试**：运行用户指定或认可的一批测试用例，其目标是证明软件达到了用户可接受的标准。
- **认证测试**：要求测试小组提供证据以表明产品或产品制造过程满足某个认证标准。

利用以上分类系统，测试人员可以判断一个测试技术的驱动因素是什么，了解它着眼于何处、又省略了哪些方面。在学习测试技术时，测试人员可以快速定位它在分类系统中的位置。在选择测试技术时，他可以同时运用来自不同分支的测试技术，以实施多样化的测试。

Cem Kaner从测试驱动因素的角度定义了测试分类系统，而测试专家Lisa Chrispin和Janet Gregory从测试技术与产品开发的角度提出了敏捷测试四象限[Chrispin09]。如图5-1所示，他们将测试技术划分到Q1~Q4 4个象限中。

- Q1：面向技术的、支持项目团队的自动化测试，例如单元测试、组件测试等。
- Q2：面向商业的、支持项目团队的自动化和手工测试，包括功能测试、样例、用户故事测试、原型、模拟等。
- Q3：面向商业的、考验产品的手工测试，包括探索式测试[①]，情景测试、可用性测试、用户验收测试、Alpha及Beta测试等。

① 我不同意Lisa Chrispin和Janet Gregory将探索式测试置于Q3象限。探索式测试是一种并行地实施测试学习、测试设计、测试执行和结果评估的测试风格。作为一种测试思维方法，它可以指导4个象限的任何一种测试技术的使用。

- Q4：面向技术的、考验产品的、使用工具的测试，例如性能测试、负载测试、安全性测试、质量特性测试等。

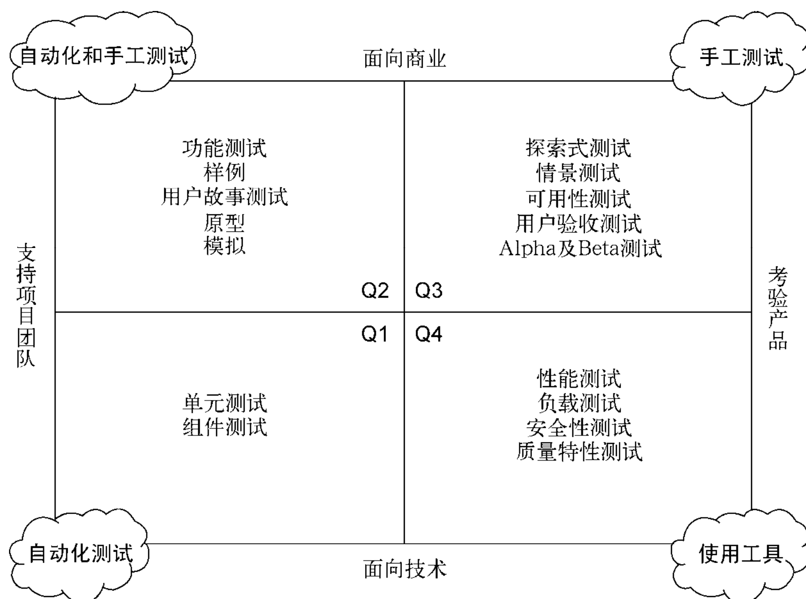


图5-1　敏捷测试四象限

利用敏捷测试四象限，测试人员可以快速理解测试技术在软件开发中的位置，并根据当前任务选择合适的测试技术。可见，掌握几种测试分类方法，可以帮助测试人员从多个角度思考测试技术，获得比较全面的理解。

5.2　启发式方法

在分析具体的测试技术之前，本节将简介启发式方法的基本概念，因为本书已经讨论或即将讨论的测试技术几乎都属于启发式方法。

启发式方法是指人们在学习、发现、解决问题时所使用的一种基于经验的方法[WikipediaHeuristic12]。在日常生活中，人们常用经验来处理复杂的现实问题。例如，去超市选购牛奶，货架上的产品可能有几十种之多。为了在几分钟之内作出决定，购买者往往会综合几条经验规则去选购。例如价格高意味着质量好、生产日期越近越好、熟悉的产品比较好（产品的熟悉程度有时取决于厂商所投放的广告）等。这些经验在大多数时候是有效的，能够快速地获得比较好的答案。但是任何经验都不能适用于所有情况，不假思索地使用它们，会导致“认知偏差”，以致于得出错误的结论。

在计算机领域，启发式方法被用于获得接近最优解的近似解。计算机领域存在大量的复杂问题，研究者对于一部分问题提出了算法，能够在可接受的时间获得它们的最优解。例如，计算机科学家Edsger Wybe Dijkstra提出的最短路径算法，以$O(n^2)$的时间复杂度计算出联通图中两点间的最短路径。对于另一些问题，并不存在多项式时间的最优解计算方法。对此，研究者们提出了启发式方法，它们基于某种简化的规则或理论，能够迅速地搜索解空间，发现足够好的解决方案。本质上，启发式方法是一种有风险的“捷径”，不能保证提供最优解。虽然在多数情况下它可以提供近似解，但是有时它会给出很差的答案。

在软件测试领域，测试人员也面临高度复杂的问题：如果穷尽测试是不可行的，如何从无穷多的测试用例中选择最有代表性的测试集合，以发现所有的软件缺陷？对于大多数软件，选择最有代表性的测试集合和发现所有缺陷是“不可能任务”，是无法在有限时间内完成的。为此，测试实践者基于思考和经验提出了一批指导测试设计、测试执行和结果评估的启发式方法。例如，边界测试的经验是软件常常在处理数值边界时出错，等价类分析的经验是等价类中的一个取值可以代表其他所有取值。实际上，启发式方法一直是测试实践的主流方法，大多数耳熟能详的测试方法都基于某些启发式的想法。测试小组通过使用多种启发式方法，可以发现大部分重要的缺陷，以构建满足市场要求的“足够好”的软件。

在测试中，启发式方法针对复杂的测试问题提出了一种简单的、较可能成功的解决思路。使用启发式方法，测试人员可以快速地采取行动，在实践中去探索答案，从而避免陷于无止境的问题分析。启发式方法是语境依赖的，即一种启发式方法会在某些环境中大获成功，却在另一些环境中不起作用。因为启发式方法有时候会失败，明智的测试人员不会只依赖一种或少数几种启发式方法，他会综合运用多个启发式方法。例如James Bach所提出的启发式测试策略模型用5个方面、几十个因素来驱动测试设计[Bach12]（参见4.2.1节），又例如The Test Eye发布的文档讨论了测试想法的37个来源[TheTestEye12]（参见3.2.8节）。此外，测试人员需要用批判性思考来研究他所使用的每项技术，知晓它的优点与不足，从而在实际测试中扬长避短 [Bach04b]。而研究技术必不可少的一环是在测试实践中使用它，通过实战来积累经验与教训。

除了收集、学习、整理已有的启发式方法，优秀的测试人员还会开发出自己的启发式方法。测试专家James Bach长期研究并实践软件测试，提出过许多启发式测试方法。他建议测试人员在测试实践中建立启发式方法，并给出了如下实施方案[Kelly05a]。

(1) 尝试去解决一个问题。

(2) 分析问题和解决方案，识别出问题的特征和解决方案的模式。

(3) 深入研究所发现的模式，理解它所针对的问题，明确它的适用范围、基本假设、实施步骤、检查方法、优势与不足，从而获得精炼的、可用于实战的启发式方法。

(4) 命名该启发式方法。许多方法的名字来自于它们的“隐喻”，例如漫游测试就用“漫游”来比喻测试过程对于产品空间的遍历访问。

(5) 在实际工作中，使用该启发式方法，并利用真实反馈来改进它。

- 发现方法的不足之处，并加以完善。
- 更改方法的名字，使名字能够揭示方法的核心特征，且容易记忆。
- 一个启发式方法要经过多次实验才能成熟。测试人员既要合理地质疑方法的有效性，又要对持续改进的结果抱有信心。

此外，好的方法不但需要反复磨炼，还需要集思广益。当新方法比较成熟时，测试人员可以把它分享给同事、测试专家或测试社区。邀请不同的人评论方法、收集来自各方面的意见，有助于发现方法的优缺点，从而更有效地改进和使用。

5.3　测试先知

所有测试都离不开测试先知，它是应用任何测试技术都需要考虑的环节。本节首先讨论测试先知的定义，然后介绍一组有帮助的测试先知。

5.3.1　测试先知的定义

测试先知是一种识别潜在问题的启发式原则或机制[Kaner10]。该定义和常见的“测试先知是判断测试是否通过的方法”有三点重要区别。

第一，测试先知的任务是“识别潜在问题”。

所谓“问题”是降低软件质量的因素，而质量是对某个（某些）人而言的价值 [Gause89]。可见，“问题”的完整含义是“某些人的问题”，是一个主观的价值判断。为了有效地识别问题，测试人员需要理解项目关系人（客户、用户、产品经理、程序员、管理人员等）对软件的期望，知晓软件通过何种途径为他们服务。在测试中，他需要用不同关系人的视角考察软件，来发掘危害软件价值的问题。如果他只从自己或单一关系人的角度考虑软件，那么他可能错过对其他人而言很重要的问题。

我发现我有时会落入一个思维陷阱，即测试时只考虑软件的行为是否符合产品经理编写的规格说明，而忽略软件设计是否真正提高了软件对用户的价值。这样，我可能放过一些符合规格说明但值得改进的问题。通常，有经验的产品经理可以提出一个“足够好”的设计，然而“优秀”的产品必须要经过实际使用的“打磨”。我发现优秀的测试人员会从用户的角度“把玩”产品，对产品的设计和实现提出很多的疑问或意见。负责任的产品经理和程序员会感谢测试人员的琢磨，并认真思考他们的提议。

第二，测试人员不能依靠单一的测试先知去判断测试是否通过。

由4.2.2节关于软件输入与输出的讨论可知，软件行为可能产生多种输出， 每种输出都可能隐藏错误。比如，测试一款计算器软件，当输入“1+1”时，软件返回结果2。从数学运算的角度，

这是正确的，但是该计算操作可能还隐藏着其他问题。例如，软件显示结果时，它的界面快速地闪烁了一下，这会引起用户的不适。再例如，软件用了2秒钟的时间才返回计算结果，这样的运算速度破坏了用户体验。再例如，每次算术运算都会泄漏1MB内存，经过多次计算后，计算器进程将占据大量的内存，以致影响到计算机性能。可见，即便是最简单的操作，也需要从多个角度进行检查。一个测试先知只检查软件的特定方面，能够发现相关的缺陷，但不能发现所有的问题。因为测试人员需要全面调查软件的质量信息，所以他应该运用多种测试先知去检查软件的行为。如果他的测试执行缺少某个重要的测试先知，那么他很可能错过一些特定方面的错误。

第三，测试先知本质是一种启发式方法，它提供了有效的检查策略，但不能发现所检查领域内的所有缺陷。

正如编程小组不能构建完美的软件，测试小组也不可能构造完美的测试先知。反过来说，如果某种方法可以提供绝对正确的结果，编程小组一定会用它来实现软件，而无需进一步测试。在真实项目中，测试先知是一种“捷径”，它基于某种原则或机制，针对特定领域提出一些测试想法，用不完美但可能成功的策略去检查隐藏的缺陷。例如，测试数学函数时，一种典型的测试先知是检查函数在某些点或某个区间的返回值是否符合其数学性质。该测试先知利用数学知识简化了测试设计，用有限的检查取代了穷尽测试。其不足是它不能检出函数值在采样点或采样区间之外的错误。总之，测试人员需要合理地质疑测试先知的有效性，并通过测试反馈来调整测试先知，使它符合产品特征并具有较强的错误识别能力。

简而言之，测试先知是识别缺陷的启发式方法。它不能判断软件是否通过测试，只能判断软件在测试过程中是否出现明显的失败。测试人员需要从不同关系人的角度，设计多个测试先知来考察软件的行为。在测试过程中，他还需要持续关注并改进测试先知的有效性。唯有如此才能避免严重的缺陷遗漏。

5.3.2 FEW HICCUPPS

FEW HICCUPPS是测试专家Michael Bolton和James Bach提出的识别和运用测试先知的想法列表[Bolton05][Bolton12]。他们建议测试人员使用以下启发式指导词来检查被测产品：

- **F**amiliarity（熟知）
- **E**xplainability（可说明性）
- **W**orld（世界）
- **H**istory（历史）
- **I**mage（远景）
- **C**omparable Products（可对比的产品）
- **C**laims（声明）
- **U**ser’s Expectations（用户期望）

- ❑ **P**roduct itself（产品本身）
- ❑ **P**urpose（意图）
- ❑ **S**tatutes and Standards（法规和标准）

指导词“熟知”建议测试人员在产品中搜索他熟悉的缺陷模式。当测试人员刚接触一个新产品时，他通常会利用已有的经验来寻找缺陷。在测试过程中，测试先知来自于典型的缺陷和以往的经验。例如，测试人员曾经使用极限值发现过许多软件缺陷，于是他在新产品中搜索可以输入数值的控件，然后尝试用最大值、最小值、零等极限值来考验软件。利用该测试策略，他可以快速地发现一批他熟悉的缺陷，并为更深入的测试积累知识和信心。不过，该测试策略可能让测试人员在很长一段时间内只关注特定类型的缺陷，而忽视了新产品特有的缺陷。人总是愿意做自己擅长的事情，但是测试要求测试人员较全面地掌握系统，并探索各个区域的细节。所以，测试人员需要交替使用多种测试策略，花一些时间寻找熟知的缺陷，然后运用其他启发式指导词去探索。

指导词“可说明性”是指产品应该容易理解，测试人员能够将它的行为向他人解释清楚。测试人员可以时常反问自己：我可以将这部分功能详细地解释给没有经验的用户吗？他会认为我的解释很合理吗？这样反思有两个好处。

第一，它让测试人员从新用户的角度考察软件的功能。有时，测试人员长期测试一款产品，对产品的行为习以为常，对一些不方便的设计拥有较高的容忍度，对一些新用户能立即发现的问题会熟视无睹。强迫自己像新用户那样思考，可以发现一些以前忽视的问题。此外，他还可以邀请其他领域的测试同事进行结对测试。在结对过程中，他仔细地解释软件的功能，结对伙伴则不停地质疑其解释的合理性。这可以在短时间内发现一批不合理的设计或实现缺陷。

第二，要求自己明确地解释软件的行为，可以暴露出自身知识上的不足。如果测试人员不能清晰地解释软件的行为，他应该感到警惕。这可能意味着他不了解业务逻辑，不知晓产品设计，不清楚实现细节。这些知识上的漏洞会导致薄弱的测试设计和严重的缺陷遗漏。不过，从积极的角度来看，困惑是一种测试工具[Kaner01]，它指引测试人员针对他不能解释的问题进行探索。通过积极地学习和测试，测试人员不但可以弥补知识的漏洞，还可以发现隐藏的缺陷。此外，如果产品经理或程序员不能清晰地解释他所负责的功能，这就暗示他们的工作可能存在盲点。测试人员可以利用该提示去仔细测试这些功能。

指导词“世界”建议测试人员用现实世界的常识去评估软件。用户使用软件去完成现实世界的任务，自然期望其行为符合一般期望或常识。例如，在线交易网站应该只显示用户信用卡的后4位，以避免信用卡被盗用。如果“客户信息”页面展示了完整的信用卡卡号，这很可能是一个缺陷。又例如，一款编辑软件要保证用户数据的安全，它要么自动保存文件，要么在退出时提示用户保存被修改的文件。如果编辑软件在某个情景中既不保存文件，也不提示用户保存文件，就径自退出，那么该行为会导致用户数据的丢失，是一个严重的缺陷。可见，常识是一个有价值的测试先知，它是篇幅有限的规格说明在复杂世界的重要补充。为了更好地利用该测试先知，测试

人员需要拓展自己的知识面，多学习与软件和业务相关的知识。否则，他会因为不了解相关知识或秉持错误的观念而错过一些缺陷。

指导词“历史”建议软件的功能应该与它的历史行为一致。许多软件发行过多个版本，当前版本的功能应该与以往版本一致，以保证用户的数据、技能、经验持续有效。例如，Microsoft Word是一个有20多年历史的软件，耳熟能详的版本包括Word 95、Word 97、Word 2000、Word 2003、Word 2007、Word 2010、Word 2013等。如果Word 2013的排版结果与历史版本有重大区别，那么它就不能妥当地呈现用Word2007、Word2010等精心编排的文档。这是千千万万的Word用户所不能接受的，是很严重的缺陷。又例如，Microsoft Word提供了对象模型（即一组面向对象风格的API），让用户和第三方的程序员可以编写VBA程序来扩展Word的功能或完成自动化任务。通常，新版本的Word会根据功能的变化来修改对象模型，但是其核心对象模型非常稳定，能保证大多数既有VBA程序可以正常地运行。如果核心对象模型的行为发生改变，那么用户的VBA程序就会报错，导致用户不能正常工作，所以这是一个严重的缺陷。

将历史版本作为测试先知时，测试人员要注意两个潜在的问题。第一，历史版本可能包含错误，使用错误的测试先知可能掩盖长期存在的缺陷。所以，测试人员要始终对历史版本抱有合理的怀疑，并用多个测试先知去评判软件的行为。如果发现一个存在于多个版本的缺陷，他应该很仔细地报告这个缺陷，写明该缺陷可能在哪些历史版本上重现，并强调修复它的价值。因为许多开发团队倾向于不修复历史版本可复现的缺陷，测试人员要充分说明它对软件价值的损害，以据理力争。第二，测试人员可能被历史版本束缚了思路，而忽视了优化设计的机会。好的设计既利用已有设计的资产，又会做出细节或整体的改进，而许多重大成功往往来自突破性的设计。例如，Word 2007放弃传统的Windows菜单界面，启用Ribbon界面。虽然一些用户抱怨Word变得陌生，但是更多的用户开始拥抱新界面。然后，Office 2010的所有程序都启用了Ribbon界面。在新界面的帮助下，Office 2010的销售量超过了所有历史版本。这说明测试人员应该关注新的应用情景和技术趋势，反思现有设计，并提出有建设性的意见。

指导词“远景”要求软件的表现符合开发团队设定的期望。例如，Google+的开发团队对产品的期望是：让用户轻松自如地在社交网络中表达自我和彼此联系，同时保护他的隐私。因此，他们决定Google+的核心价值是：社交、表现力、自如、相关、可扩展、隐私（参见3.2.2节）。在如此远景的指导下，测试团队开发出一些具体的检查规则：“只要几次点击就可以创建并加入群聊”、“只要一次点击就可以关闭视频和音频输入”、“只有被邀请的用户才能加入群聊”等。以远景为测试先知，测试人员可以将精力集中在产品最有价值的属性上，有助于发现降低产品市场竞争力的严重缺陷。

指导词“可对比的产品”建议测试人员使用相似产品作为测试先知来评测当前产品。软件的功能很少是“举世无双”的，在多数情况下，已有其他产品实现了相似的功能。通过对比相似的产品，测试人员可以较快地发现被测产品的问题。例如，测试Microsoft Word时，就可以考虑以下策略。

- **比较同一产品家族的其他应用**。例如，测试Word的图片处理时，可以对比Word、Excel和PowerPoint的相关功能，因为它们拥有几乎相同的图片处理特性集。如果在Word中调整一张图片的颜色，发现了可疑的结果，测试人员可以在Excel和PowerPoint对同一张图片实施相同的操作。如果Excel和PowerPoint的结果看上去更好，那么可以推断Word存在缺陷。
- **比较彼此竞争的产品**。例如，测试Word排版时，可以对比Word、MAC Word和OpenOffice的排版功能。让这些软件打开同一份复杂文档，比较它们所呈现的文档内容。如果Word的排版效果不如其他软件，测试人员需要提交缺陷报告，并附上评测所使用的文档和排版结果截图。
- **比较拥有相似功能的产品**。例如，测试Word的图片导入功能时，可以对比Word和Windows画图板。如果Word不能导入一张图片，但是Windows画图板却可以打开该图片，那么Word很可能存在缺陷。
- **比较功能简单、可靠性高的产品**。例如，测试Word与输入法的兼容性时，可以对比Word和Windows记事本。如果某款输入法在Word中不能输入，但是在记事本中正常工作，那么可以推测Word存在缺陷。
- **比较期望取代的产品**。例如，测试Word的电子墨水功能时，将它和传统的纸笔进行比较。测试人员可以调查：用户使用电子笔，能在Word中自如地书写吗？其用户体验能够到达纸笔的水平吗？其使用情景与纸笔相比有何优劣？
- **比较公认的“专业”产品**。测试Word 2013的“阅读视图”时，可以对比Word、Kindle for PC和Kindle for Windows 8。Kindle for PC是一款桌面软件，提供了丰富的电子书阅读功能。Kindle for Windows 8是一款Windows 8应用，面向触屏提供了流畅的阅读体验。通过对比这两款软件，测试人员可以评估Word的阅读视图在鼠标、键盘、触摸等操控方式下的表现，并分析该功能是否达到了典型阅读软件的水平。

总之，测试人员可以根据被测功能，选择一批“可对比的产品”进行比较。好的参照物不但可以提高测试效率，还能够增强缺陷报告的说服力。

指导词“声明”要求产品满足项目文档关于其功能和质量的论述。常见的项目文档包括需求文档、规格说明、用户手册、广告文宣、营销资料等。在项目过程中，许多设计决策是逐步浮现的。例如，测试人员通过实际使用产品，发现了一个设计缺陷。为了修复该缺陷，产品经理和程序员经过讨论，对产品设计进行了调整。随着此类设计变更的累积，产品可能不再满足原有文档的描述，甚至背离了设计初衷。对此，测试人员需要分析项目文档，抽取出它们对产品的声明，用作测试先知来测试软件。如果发现文档和产品不一致，他可以提交代码缺陷，也可以提交文档缺陷，要求修订文档。诸如用户手册、宣传资料等文档是最终产品的一部分，要认真对待。

指导词“用户期望”建议测试人员站在用户的角度思考，以检查软件是否真正地为用户服务，并达到了他们的期望。通常，较好的测试方法是发布软件的早期版本，邀请最终用户参与测试，

观察他们遇到的困难，并收集他们的反馈意见。此外，测试小组还可以研究用户群体，建立几个典型用户角色，来代表不同年龄、技能、特点的用户。然后针对用户角色，设计若干个他们需要完成的常见任务，并利用这批任务实施情景测试（5.6节将讨论情景测试）。在测试过程中，测试人员将自己当成用户角色，以该角色的思维方式操作软件。在测试间隙，他可以反思测试过程，并自问：软件能够帮助用户顺利完成任务么？哪些地方使用起来很不方便？哪些操作很难完成？能否调整软件，使用户可以更方便地达成目标？

指导词“产品本身”要求产品所拥有的功能和使用模式彼此一致。在一款大型产品中，一组功能是由多人设计并实现的。如果设计者之间的理念发生冲突而他们又缺乏沟通，产品的功能之间就存在不一致性，以致于让用户感到困惑。一个很知名的不一致问题是快捷键Ctrl+F在Microsoft Outlook中的作用。在阅读邮件时，Ctrl+F的行为不是大多数人期望的字符串查找，而是转发邮件。偏偏在撰写邮件时，Ctrl+F的行为又变回字符串查找。这给Outlook的用户带来了许多困扰，他们很难预测Ctrl+F的行为，也很不习惯Ctrl+F违反常识的举动。另一个例子是如图5-2所示的PowerPoint 2013的图片编辑界面，其中“图片样式”（区域A）的图标风格与“图片效果→阴影”（区域B）的图标风格不一致。实际上，区域A的图标是Office 2013的新风格，而区域B的图标继承自Office 2010。为了发现此类问题，测试人员可以使用功能漫游去游历产品的多个功能，或运行覆盖多个功能的情景测试，以发现潜在的不一致性。



图5-2 不一致的PowerPoint图标

指导词“意图”要求产品能够体察并服务于用户的意图。例如，OneNote for Windows 8[①]是一款为触屏优化的软件，图5-3是它的截图。其中，用户创建了一个列表，包含两个条目：Item1和Item2。用户将Item1的字体调整为Verdana和13磅，然后选中Item2，准备进行相同的字体调整。如图5-3所示，OneNote认为用户会重复上一次的操作，于是将字体的默认值调整为13磅，用户只要在Font Size上轻轻一点，就可以将Item2的字体尺寸调整为13磅。在大多数情况下，OneNote的这个行为符合用户的“意图”，是一项贴心的设计。但是，此时点击（或触摸）Font，OneNote却没有选中上次使用的Verdana，而是选中了Item2的当前字体Calibri，详情请参考图5-4。该行为既违反了测试先知“产品本身”，也违反了测试先知“意图”。

① 本书所测试的OneNote for Windows 8的版本是15.0.4454.1006。



图5-3　在OneNote for Windows 8中调整字体

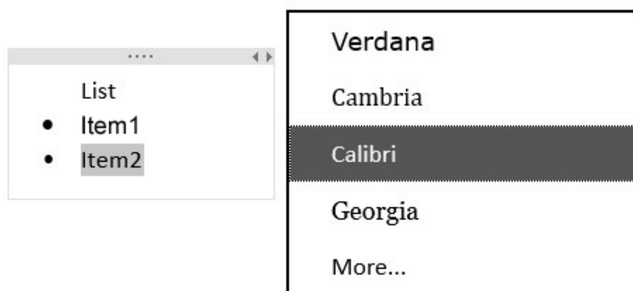


图5-4　OneNote for Windows 8显示Item2的当前字体

单纯地测试软件功能，可能不能发现此类缺陷，因为如果不考虑使用情景，诸如图5-4所示的行为并无不妥。测试人员应该用软件去完成一项有意义的且相对复杂的任务，在真实的使用语境中推敲软件的功能是否符合用户意图。例如，测试人员可以用OneNote编写一个测试小组的“新人入职指南”，覆盖文字编辑、相片处理、图表制作、列表编排、页面布局等多项功能，并确保指南内容丰富、排版美观。在测试过程中，测试人员仔细观察并分析产品的行为，可以发现一些实际用户很在意的细节问题。

指导词“法规和标准”建议测试人员研究产品相关的法令法规、行业标准、合同约定、技术标准等强制性规则，用它们作为测试先知来指导测试设计和评估。例如，微软、谷歌、苹果等公司为各自的操作系统开设了应用商店，鼓励程序员向其提交应用程序，并给予销售分成。这些公司都发布了应用开发的指导文件和应用需要满足的强制性规定。测试人员需要仔细阅读这些资料，以它们为测试先知来检查软件。许多案例表明，一些看似无碍的细节实际上触犯了应用商店的规则，并导致应用无法通过审查，延后了发布时间，阻碍了业务发展。如果测试人员可以及时发现并报告这些问题，程序员就能及时修复，产品也能顺利上架销售。为了构建平滑的发布流程，依据“法规和标准”进行测试是必不可少的一环。

FEW HICCUPPS本质上是基于一致性的启发式测试先知。测试人员利用一组启发式指导词，设计测试并检查软件在特定方面的一致性。综合使用它们，可以从不同角度考察产品，有助于更全面的思考、更周详的测试。

5.3.3　约束检查

许多软件具有复杂的计算逻辑，测试人员很难直接判断其计算结果是否正确。这时，他可以根据产品的业务逻辑和实现策略，提出一组约束规则，用相对简单的规则去发现复杂计算的错误。虽然约束规则可能会错过一些缺陷，但是它们可以用较少的资源测试复杂的产品，在测试实践中拥有重要作用。

测试专家Harry Robinson曾经在谷歌公司测试谷歌地图的驾车导航功能[Robinson10]。他需要检查谷歌地图是否正确地给出了从地点A到地点B的导航路线。这是一项有挑战的测试任务。第一，谷歌地图拥有海量的用户，他们有不同的出发地（地点A）和目的地（地点B），因此很难构建“典型”的测试输入集合。第二，导航问题在现实世界并不存在“标准答案”或“最优解”，两条不同的导航路线可能都满足用户的要求，因此测试人员很难断言某条路线一定错误或一定正确。第三，导航算法是考虑多个因素的人工智能算法，实现复杂度很高，因此重新开发一套导航算法作为测试先知即便是可行的，也会耗费大量的测试时间，给项目开发带来风险。

Robinson的策略是构建一个约束规则集，让测试程序利用这套规则去检查海量的地址组合。该测试程序的算法如代码清单5-1所示。其输入是美国所有邮政编码的配对，大约有10亿个输入，覆盖了大多数美国用户的情况。测试代码调用谷歌地图，获得每对邮政编码的导航路线集合，然后用规则集对导航路线进行检查。如果规则检查发现了一个失败的案例，程序会将失败的详细情况报告给Harry，请他做进一步的分析。

代码清单5-1　测试地图导航的算法

```
for pair_of_address in all_pairs_of_american_zip_code:
    directions = call_Google_map_for_direction(pair_of_address)
    for rule in direction_constraint_rules:
        result = rule.call_Google_map_and_check(directions)
        if result.failed:
            result.report_to_tester()
```

这是一个相对简单的测试程序，Harry用很短的时间就完成了开发。于是，工作重点就变成设计约束规则。开始时，他设计了一条规则：“从A到B的导航路线的长度与从A到B的直线距离没有显著差别。”不幸的是，测试程序在使用这条规则时产生了许多“误报”。因为，受到河流、湖泊、山脉、公园等对象的阻隔，两地之间的导航距离很可能远远大于其直线距离。

于是，Harry将规则替换为：“从A到B的导航路线的长度与从B到A的导航路线的长度没有显著差别。”该规则可以看作指导词“产品本身”的一个应用，它对比谷歌地图的两个输出来检查

功能的一致性。在实际测试中，该规则的误报率较低，且发现了一些有趣的错误。图5-5是当时Harry发现的一个错误，它显示了两条出发地和目的地对调的导航路线，左下方的路线比较合理，而右上方的路线则明显有误。

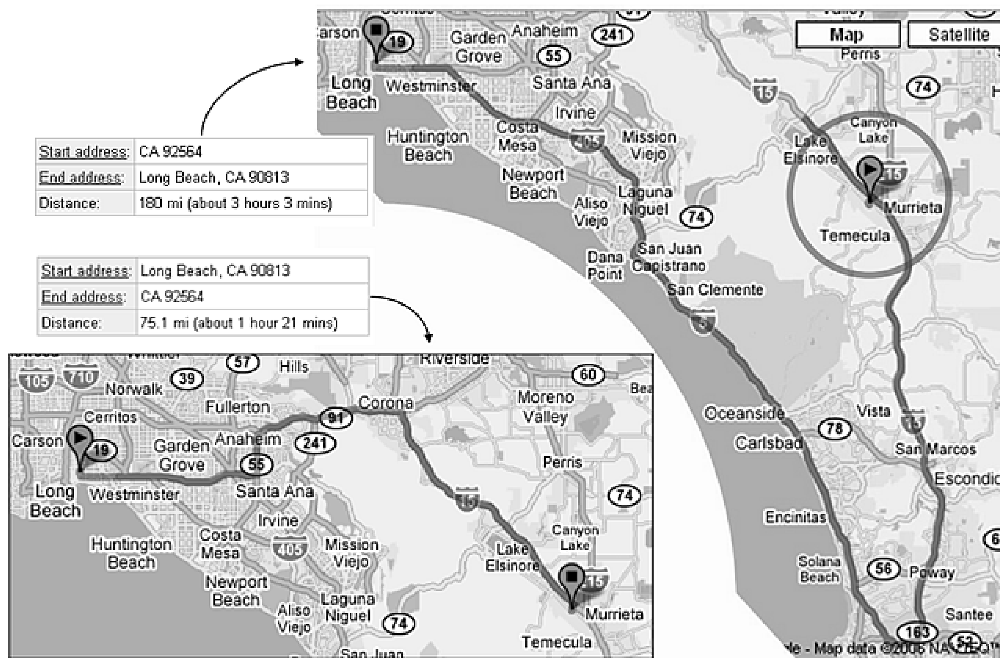


图5-5　谷歌地图给出了不一致的导航路线

分析Harry的测试策略，可以总结出一些实施约束检查的经验。

- 用简单的程序去测试复杂的产品。简单的测试程序更容易开发，能够快速投入测试，且可以随着产品发展随时调整。
- 用海量的数据和启发式规则去发现失败案例，让测试人员做进一步调查。
- 测试人员用较少的时间构造测试程序，用较多的时间设计约束规则，分析失败案例，改进约束规则。
- 测试先知的构造不是一蹴而就的。在测试迭代中，测试人员需要根据测试反馈，修改、替换、增加约束规则，以逐步完善测试先知。
- 当约束规则集不能发现失败案例时，测试人员要考虑增加新的规则。例如，他可以尝试新规则："从地点A到地点B的路线上选择一个点C，调用地图获得A到C的导航路线和C到B的导航路线。A到C的路线长度加上C到B的路线长度应该约等于A到B的路线长度。"

软件产品是多种多样的，测试人员可以从多个角度设计约束规则。以下是一些典型的测试设计策略。

策略1：基于"数学性质"的约束规则利用数学知识构造检查规则

假设，测试人员需要测试数学函数double System.Math.Sin(double x)，但他暂时没有参考程序或其他测试先知。为了快速地发现严重的缺陷，他可以考虑以下约束规则，做一些快速的测试。

- 在数学上，sin(x)对于一些输入值有非常明确的结果，例如：sin(0) = 0，sin(0.5 × π) = 1，sin(π) = 0，sin(1.5 × π) = −1，sin(2 × π) = 0。测试人员用这些输入值测试函数，检查返回值应该等于或非常接近目标值。
- 在数学上，sin(x)在区间[0, π]中单调上升。测试人员可以在此区间取一些点，检查相应的计算结果符合单调上升的特性。
- 在区间[0, 2 × π]中均匀地取200个点，调用被测函数获得相应的函数值，然后用软件绘制出函数值的变化曲线，考察该曲线是否符合sin(x)的基本特征。图5-6是我用Microsoft Excel绘制的函数System.Math.Sin的散点图，因为取值点很密集，所以函数值连成了一条曲线。虽然不能精确判断，但看得出该曲线符合sin(x)的基本特性。

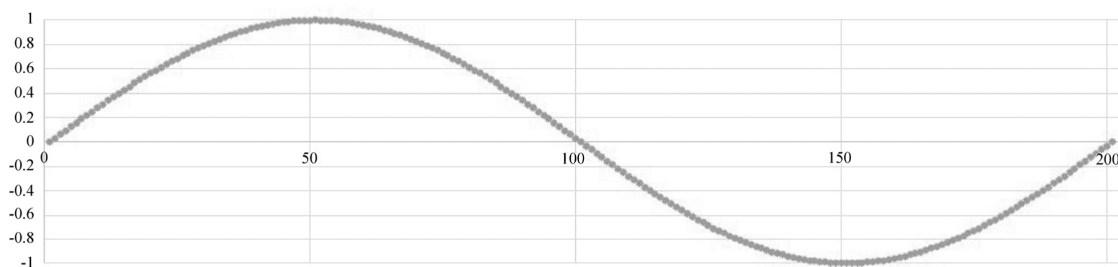


图5-6　double sin(double x)的变化曲线

虽然以上规则不能保证被测函数一定正确，但是它们可以快速地发现明显的问题，用很少的时间获得基本的质量信息。基于测试结果，测试人员可以更好地设计下一步的测试。

策略2：基于"统计特性"的约束规则利用被测对象的统计特性构造检查规则

例如，测试人员要测试函数double System.Random.Sample()能够随机地返回0.0和1.0之间的数，他可以考虑如下约束规则。

- 首先，随机数应该均匀的分散在区间(0.0, 1.0)中。测试人员可以调用System.Random.Sample()一万次，然后统计落入(0.0, 0.1], (0.1, 0.2], (0.2, 0.3], …, (0.9, 1.0)这10个区间的随机数的个数，这也检查了被测函数并没有返回区间(0.0, 1.0)之外的数值。他可以绘制出类似于图5-7的随机数分布图，来检查其分布是否均匀。在图5-7中，10万个随机数大致均匀地分布在10个区间中。
- 其次，随机数的生成应该没有明显的规律，即不能利用以往生成的随机数来预测下一个随机数。测试人员可以多次调用被测函数，绘制出类似于图5-8的随机数散点图，来检查

该规则。在图5-8中，随机数的数值与函数调用次数、之前的随机数、之后的随机数没有明显的关系。

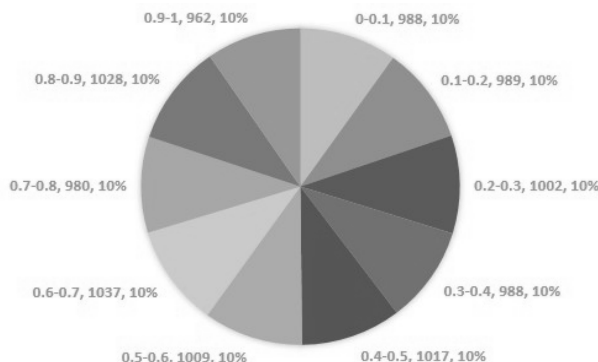


图5-7　随机数在10个区间的分布

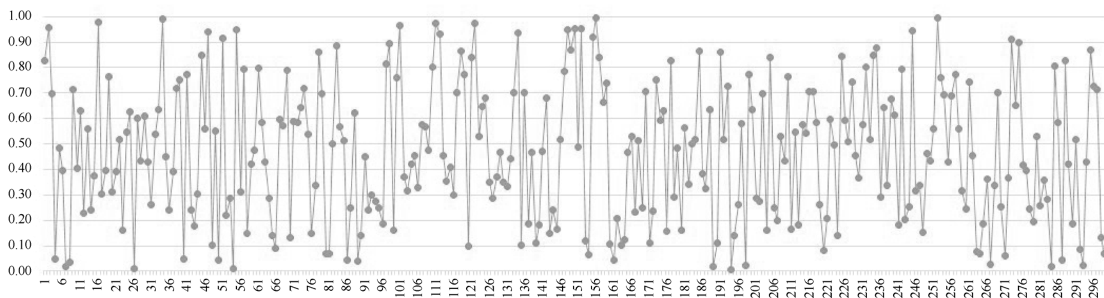


图5-8　随机数的散点图

测试人员还可以用更严格的规则来测试随机数函数。对于随机性要求不高的领域，图5-7和图5-8的测试结果表明随机数函数并没有明显的问题。

策略3：基于“格式”的约束规则检查数据符合特定的格式

现实世界的许多数据（如电话号码、邮政编码、日期、电邮地址、IP地址、文件格式等）都拥有明确格式要求。测试人员可以参考这些格式，设计约束规则去检查软件产生的数据。例如，Windows事件查看器显示了操作系统记录的事件和事件发生的时间。假设当前系统的地区设置是“美国”，那么它显示的时间字符串的格式应该类似于“12/21/2012”。如果它显示的时间字符串是欧洲格式，类似于“21/12/2012”，那么其日期格式化功能存在缺陷。再例如，许多程序会生成docx格式的Word文档，以实现办公自动化。测试文档正确性的第一步是验证其格式。docx格式本质是一个压缩文件，包含了一批XML文档和数据文件。微软提供的Open XML SDK[①]可以检查一个docx

① http://www.microsoft.com/en-gb/download/details.aspx?id=30425。

文件是否被正确压缩，其包含的XML文档是否符合正确的XML架构。于是，测试人员可以编写一个测试程序，调用Open XML SDK来检查所生成文档的格式。

策略4：基于“业务知识”的约束规则使用领域知识来设计检查规则

例如，Harry Robinson根据路线导航的常识，提出了一组规则来检查谷歌地图的导航结果。又例如，我曾经测试过一个订单处理系统，其输入是一组数量庞大的订单，每个订单有独立的编号，其字段也有独立的编号。系统分析每个订单的字段，给出相应的计算结果，该结果包含原订单号或字段号。根据系统的业务需求，我提出了“编号守恒定律”，即所有输入的订单号和字段号应该出现在输出结果中，且输出结果包含的编号必须存在于输入数据中。我用代码实现了该约束规则（和其他一批规则）的自动检查，将其作为自动化系统测试的测试先知。在每天执行的自动化测试中，这批规则发现了许多现因为代码改动而引发的错误，取得了预期的检测效果。

总而言之，测试先知的类型是多种多样的，每种类型都有自己的优点和不足。测试人员需要透彻地理解测试先知的理论和方法，知晓它能发现哪些类型的错误、不能发现哪些类型的错误。然后，从不同关系人的角度，使用多个测试先知去检查软件的行为，让它们互为补充、彼此增强。因为测试先知可能存在缺漏，他应该在测试迭代中持续评估测试先知的有效性，通过完善并补充测试先知，来稳步提高测试检查的广度和深度。

5.4 漫游测试

漫游测试是在特定主题指导下对产品进行探索的一组测试方法。不同的漫游测试方法有不同的主题，侧重于探索产品的不同方面，其共同的特征是测试人员会游历产品空间，以尽力覆盖某类对象或发现某类错误。

通常，漫游测试的名字会反映其主题，一般表现为待测对象、测试策略或测试隐喻。例如，功能漫游建议测试人员去探索一组相关的功能；用户漫游建议测试人员从用户的角度考察产品功能；出租车漫游以“出租车”作为隐喻，建议测试人员从不同的途径访问同一个功能。在主题的指引下，测试人员产生一组测试想法，并在探索的过程中产生更多的想法。因此，Cem Kaner认为漫游测试是一种结构化的头脑风暴方法 [Kaner11]。

漫游测试的重心是探索，而探索的核心价值之一是学习。在探索过程中，测试人员不但可以发现缺陷，还可以深入学习软件的业务规则和实现细节，从而提出更高效的测试策略。一些有经验的测试人员会在拟定详细测试计划之前，安排几个漫游测试的测程，收集测试计划需要的信息。这几次漫游测试的重点不是发掘缺陷，而是从测试的角度研究软件，其主要产出不是缺陷报告，而是一组测试支持材料，包括软件的功能列表、风险列表、测试想法列表、需要覆盖的测试目标等。基于这些信息，测试人员可以更好地规划测试，选择有针对性的测试技术。

5.4.1　基本漫游方法

自1990年以来，James Bach、Elisabeth Hendrickson、Michael Kelly等提出了一批具体的漫游测试方法[Kaner11][Bolton09b][Kelly05b]。基本上，这是一组基于覆盖的测试技术，其测试过程需要覆盖一组预定的测试目标。表5-1总结了这组方法的特征。

表5-1　典型的漫游测试方法

方法名	覆盖对象	测试手段
功能漫游	软件的功能	功能漫游发现软件的功能区域（功能列表的主干）和细节功能（功能列表的分支）。在测试之初，测试人员通过功能漫游了解软件的整体情况，建立功能列表的主干。在测试特定功能时，测试人员用功能漫游描绘出功能列表的分支，详细地覆盖一个功能的细节（以及与它协作的其他功能）
菜单和窗口漫游	菜单和窗口	菜单和窗口漫游是一种特殊的功能漫游，旨在发现所有的菜单（包含右键点击所触发的菜单）、菜单项、窗口（包含对话框）、工具条、命令图标和其他界面控件
鼠标和键盘漫游	鼠标和键盘可以触发的功能	鼠标和键盘漫游是一种特殊的功能漫游，旨在发现所有鼠标和键盘可以触发的功能。测试人员需要尝试左键点击、左键拖曳、右键点击、右键拖曳、中键点击、滚轮滚动等鼠标操作，并尝试各种键盘命令，包含F1~F12功能键、Esc、Shift、Ctrl、Alt等。此外，测试人员还需要覆盖鼠标与键盘的协作，例如他可以探索在左键拖曳时，按下Esc键会发生什么
业务漫游	软件的业务	业务是软件需要完成的一项完整的任务，通常会涉及多个具体的功能。业务漫游需要覆盖软件支持的所有业务
错误消息漫游	错误消息	错误消息漫游覆盖软件所有的错误消息。测试人员可以询问程序员或扫描源代码，来获得需要覆盖的错误消息。此外，测试人员还可以列出他认为软件可能报告的错误，然后尽力去触发。该漫游旨在发掘软件可能遇到的异常情况，从而发现软件在异常处理方面的缺陷。这说明，虽然许多漫游测试是基于覆盖的测试技术，它们也关注着软件的风险（即可能存在的错误），并通过测试覆盖来暴露风险
变量漫游	软件的变量	变量漫游发现软件输入和输出的变量。关于变量的更多讨论请参考4.2.2节
数据漫游	软件的数据元素	数据漫游发现软件的数据元素（HTSM→产品元素→数据元素），即软件所接纳、处理、产生的数据
采样数据漫游	数据的数据元素	在了解了软件的数据元素后，测试人员选择一些复杂的、极端的数据去测试软件
文件漫游	软件所使用的文件	文件漫游发现软件所使用的文件，包括可执行文件（.exe）、动态链接库（.dll）、配置文件（.ini）、数据文件（.dat）、帮助文件（.hlp）等
结构漫游	软件的结构元素	结构漫游发现软件的结构元素（HTSM→产品元素→结构元素），即构成软件的实体，包括代码、数据、接口、文档、硬件、相关网络服务等
操作模式漫游	软件的状态	操作模式漫游发现所有影响软件行为的操作模式，这通常意味着建立软件的状态机模型。关于状态机的更多讨论请参考4.2.5节
序列漫游	软件的操作序列	序列漫游覆盖软件的操作序列，这通常意味着覆盖状态机的多条路径，每条路径都覆盖多个状态和变迁

（续）

方法名	覆盖对象	测试手段
声明漫游	关系人对软件的声明	声明漫游发现项目关系人对软件的所有声明。测试人员需要研究显式的规格说明（用户手册、帮助文件、使用教程、广告文宣等）和隐式的规格说明（关系人的邮件、口头表态、会议纪要等）。该漫游有助于建立一致性测试先知“声明”
文档漫游	软件的用户文档	文档漫游根据软件的用户手册、在线文档、使用教程来覆盖软件的功能。测试人员依据用户文档的指示来操作软件，并记录软件与文档的不一致之处
价值漫游	软件的价值	价值漫游发现软件为用户提供的最有价值的功能，从而理解软件的目标和项目团队的使命。该漫游有助于建立一致性测试先知“远景”
市场处境漫游	市场处境	市场处境漫游发现软件的目标市场的重要元素，包括竞争产品、竞争产品的特点和优势、产品用户、用户购买时所考虑的因素等。该漫游有助于建立一致性测试先知“可对比的产品”
用户漫游	典型的用户情景	用户漫游覆盖典型的用户情景。测试人员建立几个典型的用户角色，分析他们的期望和使用模式，确定典型的用户情景和任务，并予以测试。该漫游有助于建立一致性测试先知“用户期望”。5.6将介绍情景测试
配置漫游	软件的配置	配置漫游发现影响产品行为的软件设置、操作系统设置、硬件设置等配置信息
互操作漫游	与软件协作的其他程序	互操作漫游发现与软件一起协同工作的程序和服务，以及软件与它们协作所使用的接口和协议
兼容性漫游	软件的运行平台	兼容性漫游覆盖软件的运行平台，包括计算平台、操作系统、硬件设备等。对于一些手机应用，它支持数十种手机型号，做完全的兼容性漫游是不现实的。测试人员需要拟定一个设备选择方案，用手机的市场占有率、影响、用户特点等因素来选择少数手机进行测试。制定该选择方案的过程也是了解目标市场和产品价值的学习过程
可测试性漫游	支持测试的软件功能	可测试性漫游发现软件中支持测试的功能，例如日志文件记录了软件的运行情况、软件的编程接口提供了它的内部状态等。该漫游还可以发现软件在可测试性上的不足，并提出一组测试人员希望软件提供的测试辅助功能
风险识别漫游	软件的风险	风险识别漫游产生软件的风险列表。测试人员可以分析软件的各个部分，想象它可能遭遇的失败，以列出一组软件面临的风险。测试人员还可以漫游产品的功能，并沿路思考：“此处，软件可能发生什么错误？它可能遇到什么异常情况？”，以获得一组潜在风险
极限值漫游	软件的极限值	极限值漫游是一种特殊的风险识别漫游，它通过输入极限值或制造软件的极限状态来发现软件的错误。常见的极限值包括零、最小值、最大值、空值、NULL、负值（当软件期望用户输入正数时）、不正确的数据格式等
复杂性漫游	软件的复杂情景	复杂性漫游覆盖高度复杂的用户情景，这些情景往往拥有漫长的操作序列、大量的用户输入、繁复的功能组合。该漫游有助于发现软件在处理复杂情况时暴露的不足和缺陷

5.4.2 基于旅行者隐喻的漫游方法

测试专家James Whittaker基于系统化错误猜测，以“旅行者”为核心隐喻，提出了一组测试

方法[Whittaker09]。他的方法往往关注特定的风险和缺陷，是一组基于风险的测试技术。表5-2归纳了这组方法的要点，并用阴影隔出了相关的测试方法。

表5-2　James Whittaker的漫游测试方法

方法名	隐　喻	测试手段
指南漫游	旅行者遵循旅游手册来漫游景点	测试人员遵循用户手册的操作指令来使用软件。该漫游有助于发现手册与软件的不一致性
博客漫游	一些旅行者会写博客来点评城市或景点	测试人员利用第三方博客、用户论坛等资源来收集用户反馈，根据他们的建议来测试软件。这有助于测试人员按照用户的使用方式来测试软件
评论者漫游	一些旅行者会以专家的身份批评景点，指出其不足	测试人员收集批评者的意见，根据他们的负面评论来测试软件。这有助于发现不良设计对软件价值的损害
竞争者漫游	参考旅行者对其他景点的评论来改进当前城市的旅游品质	测试人员从论坛、博客、新闻、专栏等处收集用户对竞争产品的看法和建议。利用这些信息设计测试情景。这有助于评估当前产品是否满足用户需求，并发现设计的不足之处——这些问题会阻碍用户从竞争产品迁移到当前产品
卖点漫游	旅行者总是访问城市中最知名的景点	测试人员发现并测试软件的卖点，这通常是销售人员和广告资料重点强调的特性。该漫游与之前介绍的价值漫游都有助于分析软件的核心价值是否存在风险
地标漫游	旅行者在地图上标注地标，然后从一个地标走到另一个地标，直至到达目的地	测试人员选择一组相关的功能，依次测试，直到测试了所有功能。该漫游与功能漫游相似，都可以探索并学习软件的功能，用一组地标（功能列表的主干）产生详细的地图（功能列表的分支）
停车场漫游	旅行者在一处景点关门后才到达，只好在停车场打转，并寻找另一处可玩的景点	该漫游探测软件的“地形”，其主要目标是发现所有功能的入口（面向覆盖）和有可能发生的问题（面向风险），次要目标是确定具体的漫游方法可以应用的地方，幸运目标是发现一些极端严重的缺陷
知识分子漫游	有些旅行者拥有丰富的知识，其见识超过了导游	测试人员运用业务和技术知识向软件提出极具挑战的问题，例如：“软件能处理的最大流量是多少？哪些操作会索取最多的内存？”成功运用该漫游的前提是测试人员深刻地理解业务、设计和实现，并可以设计出有挑战性的测试用例
找茬漫游	一些旅行者很挑剔，会打断导游的介绍，询问一些挑衅的问题	测试人员在测试软件时，以找茬的心态去挑战软件。他提出各种问题去质疑软件，例如：“如果我这么做，会如何？如果我偏不这么做，又如何？如果我不想这么做，有什么替代方案？”该漫游有助于发现程序员设计的不足和软件逻辑的漏洞
傲慢美佬漫游	许多美国旅行者给人的印象是傲慢自大、无理取闹、常当众出丑	测试人员用一些不合“正常”逻辑的情景来测试软件。例如，测试在线交易网站，提交订单并支付后，又取消订单并要求退款。又例如，提交订单后，又申请去更改送货地点和送货方式。该漫游有助于发现软件在不常见情景中所暴露的错误
深夜漫游	一些景点和商店会通宵运营	测试人员测试一些在深夜运行的任务，例如每日业务结算、数据备份、垃圾清理等
清晨漫游	在清晨时，一些清洁工会清理城市	测试人员测试软件的启动过程，关注软件设置、操作系统配置、硬件参数等对启动的影响

（续）

方法名	隐　喻	测试手段
快递漫游	联邦快递会携带货物在城市间穿梭	测试人员跟随一组数据走遍软件的功能。例如，测试PowerPoint图片时，测试人员跟随一组图片，测试图片导入、图片编辑、图文混排、图片导出、幻灯片打印等功能，并时刻检查图片操作的正确性
垃圾车漫游	垃圾车会途经每一个垃圾箱，并做短暂的停留	测试人员选择一组目标，如所有的菜单项、所有的错误消息或所有的对话框，然后依次访问它们，并对每个目标做快速地测试
出租车漫游	出租车司机非常了解城市，能从多条路线到达目的地	测试人员发现并测试触发某一功能的所有途径，或完成某项任务的所有方式。该漫游有助于完整地考虑并测试一个功能或情景
出租车禁区漫游	出租车司机有时会尝试多种路线，以到达看似无法进入的区域	测试人员瞄准一个很难达到的软件状态，尝试各种方法去驱动软件抵达该状态。该漫游有助于完整地考虑并测试一个功能或情景
恶邻漫游	城市中有脏乱差的社区。旅行者会避之不及，侦探却要勇往直前	测试人员分析软件的哪些模块拥有许多缺陷。因为缺陷常常聚集出现，而且缺陷修复也可能引入新的问题，所以那些模块可能还存在更多缺陷。测试人员需要花时间专门测试这些模块
博物馆漫游	博物馆保存了许多历史久远的古董	测试人员软件实现，发现哪些模块拥有历史代码。当这些历史代码被修改，或应用在新环境中，它们很可能导致软件故障。因此，测试人员需要花时间测试历史代码以及它们和新代码的交互
先前版本漫游	城市总是在原有基础上发展而来	测试人员细致地比较当前版本和先前版本的行为。该测试有助于发现严重的兼容性缺陷
配角漫游	旅行者会访问著名景点，也会游历到周边地区。它们构成了城市的整体印象	测试人员重点测试软件的非主要功能。例如，对话框上有一条指向帮助文档的链接，很少有用户会注意到它。测试人员则需要点击该链接，以检查它指向正确的文档。该漫游有助于完整地测试软件的每个功能
后巷漫游	旅行者通常不会进入阴暗的后巷，但侦探恰恰相反	测试人员重点测试很少人使用的功能，以检查软件是否可以满足少数用户的特殊需求
混合方向漫游	旅行者既访问华丽的景点，又进入阴暗的后巷	测试人员将受欢迎的功能和无人问津的功能放在一起测试。这有助于发现一些隐藏很深的集成缺陷
通宵漫游	有些旅行者会在酒吧通宵狂欢	测试人员长时间用各种方法测试产品，且不重新启动它。该漫游有助于发现一些长时间运行才能暴露的缺陷，如计算误差累积、资源缓慢泄漏等
收藏家漫游	一些旅行者会收集所到城市的纪念品，收集的越多，他们就越开心	测试人员通过测试去收集软件的输出，收集得越多越好。例如，测试PowerPoint图片时，测试人员要收集各种图片导出的结果，以覆盖图片格式、图片色彩、图片效果、图片尺寸等因素。该漫游有助于周密地覆盖软件的计算结果
超模漫游	超级名模靓丽的外表会吸引最多的目光	测试人员重点测试软件界面，检查界面是否美观、控件是否正确、动画是否流畅、色彩是否协调、刷新是否及时、字体是否优雅等
孤单商旅漫游	一个商业旅行者常常访问新的城市。为了了解陌生的城市，他会选择一个距离办公地点较远的旅馆，以便他有机会在城市中穿行	测试人员选择一个功能，然后构造最长的路径去访问它，其主旨是在达到目的地之前使用尽可能多的功能。这有助于测试人员更好地理解软件，并发现多个功能交互引发的缺陷

（续）

方法名	隐　喻	测试手段
测一送一漫游	购物者都喜欢买一送一	在测试时，测试人员启动被测软件的多个进程，让它们都开始工作，并试着让它们访问相同的资源（如本地文件、网络服务等）。这有助于发现进程相互影响而导致的软件错误
苏格兰酒吧漫游	苏格兰旅行者喜欢成群结队的泡吧、喝酒、聊天	测试人员需要理解产品的用户社区，他可以参与用户讨论组，回答用户问题，阅读有关产品的博客、评测和新闻
多元文化漫游	一个国际化大都会往往拥有不同文化的社区	测试人员重点测试软件的国际化和本地化特性，例如，软件能否正确地显示多种语言的文字（包括从右向左的阿拉伯文），软件能否根据系统设置正确地显示货币、日期、数字等格式，软件能否正确地处理多种语言的输入等
阵雨漫游	遇到阵雨时，旅行者会暂停旅程。当雨过天晴，他会继续上路	测试人员启动一个耗时较长的操作，然后取消它。不正确的取消逻辑会破坏软件的状态，导致软件故障，所以测试人员要检查软件在取消之后是否可以正常工作。此外，测试人员还可以启动一个操作，不等它结束，又开始一个相同的操作。有些软件不能正确处理并发计算，两个同时进行的计算可能触发软件故障
沙发土豆漫游	有些人很少旅行，只是整天陷在沙发里看电视	测试人员尽可能不提供或修改数据，这意味着接受软件提供的默认值、保持表单字段为空、或只提交最少的数据。该漫游检查软件可以处理它提供的默认值、空值和可选字段
破坏者漫游	某些旅行者会破坏环境，并以此为乐	测试人员实施故障注入和错误容忍方面的测试。他故意破坏或移除软件操作需要的资源，然后强制软件执行注定导致失败的操作。该漫游将测试软件能否合理地处理（不可避免的）失败，且不会导致更严重的后果（如用户数据丢失）
反叛漫游	有些人总是充满反叛精神，并用实际行动反对周遭的一切	测试人员故意与软件做对，想尽办法让软件难堪。该漫游有助于检查软件能否合理地处理困难的任务和恶意的输入
敌对漫游	有些人用敌对的态度来处理他不喜欢的人或事	该漫游是一种特殊的反叛漫游，它要求测试人员输入无效的数值来考验软件
犯罪者漫游	罪犯会实施猖狂且无孔不入的攻击	该漫游是一种特殊的反叛漫游，它要求测试人员扮演黑客的角色，用各种方法和工具攻击软件
歧路漫游	旅行者会走错路，被迫掉头再尝试	该漫游是一种特殊的反叛漫游，它要求测试人员以错误的方式或顺序来操作软件。例如，测试人员故意调换工作流的步骤，故意使用未经初始化的数据等
强迫症漫游	强迫症患者会反复做同一件事情	测试人员反复执行同样的一组操作，这些操作的累积效应可能导致软件故障
混票漫游	有些旅行者会混入他人的旅行团，享受其服务	该漫游针对情景测试，以引入更多的测试变化。它要求测试人员跟随一个测试情景访问某个功能，然后再利用另一个测试情景去测试其他功能。这有助于混合测试多个情景，发现由功能交互引发的问题

5.4.3 移动测试漫游方法

测试专家Jonathan Kohl针对运行在移动设备上的软件，提出了一些移动测试漫游方法[Kohl12]。其主要特点是选择一组用户的使用情景或移动设备的特性（触屏、光线感应器、加速感应器、GPS、无线网络等），基于它们测试移动软件的功能。部分方法无法在办公室内完成，需要测试人员在室内和户外真正“移动”起来。表5-3概述了这些方法的特点。

表5-3 Jonathan Kohl的移动漫游测试方法

方法名	覆盖对象	测试手段
手势漫游	触摸手势	测试人员访问软件的每一个界面，并尝试各种触摸手势，包括轻敲、两次轻敲、按压、按压并拖动、滑动、旋转、缩放等。测试人员检查软件支持哪些手势、这些手势是否足够、支持的手势是否被正确地实现、不支持的手势是否被妥当地忽略等
方向漫游	屏幕方向	测试人员访问软件的每一个界面，并旋转设备以改变屏幕方向：从纵向视图变为横向视图，再变回来。测试人员检查两种视图是否一致、某个视图是否限制了用户的使用等
转向漫游	用户导航	用户在使用软件时很可能会改变主意，从当前界面跳转到其他界面，甚至退出程序。测试人员需要系统性地漫游软件，以覆盖导航操作，包括：退回到之前的界面再进入、取消当前操作、回到软件的主界面、退出到系统的开始屏幕再返回、关闭软件再打开、切换到其他软件再返回等。测试人员检查这些操作是否会导致数据丢失、功能异常等问题
配件漫游	移动设备的配件	移动设备常见的配件包括：耳机、蓝牙耳机、麦克风、充电器、无线充电板、外置键盘、蓝牙键盘、外置底座等。测试人员测试软件能否和这些设备很好地协作，并关注配件的接入和拔出。例如，测试一款录音软件时，测试人员需要测试设备的内战麦克风和外置麦克风。当录音还在进行中，拔出外置麦克风，这时软件会自动切换到内置麦克风继续录音吗？此时，再接入外置麦克风，软件会切换回外置麦克风吗
动作漫游	用户动作	许多时候用户会移动设备，测试人员需要漫游软件，以检查软件可以正确处理加速感应器、重力感应器等元件发出的信号，不产生让用户感到意外的情况。常见的动作包括：拿起或放下设备、拿着设备挥动手臂、边走边用、疾行时猛停、将设备丢向沙发等
光线漫游	光线情况	用户会在不同光线的环境中使用软件，测试人员需要漫游软件，以检查软件是否可以正确处理光线感应器发出的信号，并清晰地显示内容。常见的测试环境包括：日光灯照亮的房间、漆黑的房间、从漆黑的房间走入明亮的房间（或相反）、在温和灯光的阴影下、在温和日光下、在猛烈的日光下、在树荫下、在黑夜中、在黑夜的路灯下等
地点漫游	使用软件的地点	移动设备会依赖无线网络信号和GPS信号，而信号强弱与位置紧密相关。测试人员需要考察软件在不同地点的表现。常见的测试方法包括：在移动中（步行或坐车）使用软件、在高楼下使用软件（高楼对信号有干扰）、在大楼内的不同地点使用软件、穿过大楼或隧道等阻断信号的地点等
连通性漫游	无线网络的状态	测试人员用各种网络情况测试需要通信的功能。常见的测试情景包括：各种强度的Wi-Fi网络、各种强度的3G网络、从信号强的Wi-Fi网络移动到信号弱的网络、从Wi-Fi网络移动的3G网络、从有信号移动到无信号、从无信号移动到有信号等。测试人员检查软件在各种网络情况下的表现，并检查它是否可以正确地处理网络变迁

（续）

方法名	覆盖对象	测试手段
天气漫游	使用软件时的天气	天气会影响移动设备的定位服务、光线感应器、无线通信等特性。测试人员漫游软件，以检查各种温度和光线对软件的影响，常见的测试场景包括：寒冷的冬日（用户穿戴的手套会影响到触摸操作）、阳光暴晒的夏日（高温可能会影响设备的稳定性）等
对比漫游	软件的不同版本	测试人员对比移动软件在不同操作系统上的版本，如iPhone、iPad、Android、Windows Phone、Windows 8等。如果某个版本缺少一项其他版本具有的常见功能，测试人员应该报告缺陷
声音漫游	与声音有关的事件	测试人员测试软件能够正确地支持音频输出（播放声音、音乐等）与音频输入（录音、语音识别等）。此外，他还要测试软件是否可以正确处理来自操作系统和其他软件的声音中断。例如，软件正在播放音乐时，日程安排软件弹出会议通知并伴随提示音，音乐在被中断之后可以继续播放吗？如果在播放音乐时，有电话拨入，在通话结束后音乐可以继续播放吗
组合漫游	移动技术的组合	移动设备会使用多种技术，测试人员需要漫游软件，以检查软件是否可以正确地同时使用这些技术。例如，测试人员一边走一边使用软件（让GPS和加速感应器同时工作），同时让它使用互联网资源（使用无线网络），并播放音乐（使用扬声器），还使用各种触摸手势（使用触摸屏）来改变软件的状态
一致性漫游	操作系统的软件开发指南	测试人员访问移动软件的每一个界面，将软件的设计与操作系统的软件开发指南进行比较。如果发现当前软件与推荐设计不一致，测试人员需要报告缺陷
用户漫游	软件的用户	邀请最终用户试用软件，观察他们的行为，收集他们的反馈。对于小型开发团队，可以邀请开发者的家人和朋友来参与。此外，测试人员还可以拟定几个典型的用户角色，然后扮演他们，以他们的方式来使用软件。这有助于测试人员摆脱自身的使用习惯，用新眼光发现以往忽视的问题
休息室漫游	使用软件的地点	当用户在家时，他们很可能躺在休息室的沙发或床上使用软件。测试人员需要检查当用户躺下或半躺下时，他可以舒适地使用软件
低电量漫游	设备电量	当设备的电量即将耗尽时，操作系统很可能会关闭一些（软件所依赖的）服务并降低CPU的运行速度。测试人员需要漫游软件，以检查软件是否可以正确地处理这种情况。如果软件所依赖的服务被关闭，它应该以用户友好的方式报告当前情况。如果设备连上电源，相关服务被启动，软件应该恢复正常工作
温度漫游	使用环境的温度	极寒和极热的使用环境会影响移动设备的硬件表现，有可能导致软件出现故障。测试人员需要在这些环境中漫游软件，以检查软件是否出现计算错误、性能迟缓、使用不便等问题
多屏幕漫游	与其他设备的协作	越来越多的用户用多个设备来协同工作。例如，他在手机上发现了一款衣服，然后用平板电脑下单购买。又例如，他在桌面电脑上发现了一本电子书，然后使用手机下载并阅读，接着又在平板电脑上继续阅读。测试人员测试软件可以将信息方便地分享给其他设备，并接受来自其他设备的信息。他使用多款设备，检查它们是否可以组成平滑的工作流来完成同一项任务
资源漫游	软件的性能	测试人员给软件施加压力，让它全力运转或面临资源压力，以检查软件的性能问题和可靠性问题。常见的测试方法包括：同时运行多款软件、让软件读取空间耗尽的存储卡、让软件处理尺寸巨大的图片、让软件传输大量数据、在低速网络中使用软件、在低电量时使用软件等

（续）

方法名	覆盖对象	测试手段
情绪漫游	使用软件的心情	测试人员漫游软件，识别由软件引发的快乐、自豪、滑稽、疑惑、恼怒等情绪。情绪漫游关注软件可用性，关注软件带给用户的感觉，这对于移动软件非常重要。用户会毫不犹疑地删除一款让他感到被愚弄或被激怒的软件，整个过程只要两秒钟。更糟糕的是，他可能在应用商店留下1颗星的恶评，这对移动软件的推广是很不利的。面对大量的同类软件，用户在选择时会默认排除3颗星以下的软件，这意味着1颗星的软件几乎没有机会被安装。开发团队可能需要多次发布才能将用户评分拉升到3颗星或更高，而这只是为了弥补一次糟糕发布的恶果
急迫漫游	使用软件的心情	用户有时会在焦急的情绪下使用软件，例如他上班迟到，一边小跑着一边用手机向领导发邮件："今早的会议我会晚到一会。"测试人员需要模仿这种情景，用急促的动作和不耐烦的心态来操作软件，以发现软件的不足

随着智能手机成为人们强烈依赖的联络工具和计算平台，智能手机厂商加快了竞争步伐，市场上的新产品层出不穷。一些团体和个人会对新产品进行评测，并在网络上用文字、图片、视频等形式公布测评结果。对比Jonathan Kohl的移动漫游测试，不难看出有一些评测从移动测试的角度存在以下不足。

- **这些评测的主要形式是评测者坐在室内把玩手机，只包含很有限的移动场景。但是，用户使用手机的情景是丰富多样的。**他们会躺着使用、边走边用、坐在高速行驶的车辆中使用、在不同光线环境中使用、在烈日下使用、在黑夜中使用、在各种无线网络中漫游……有限的评测场景不能全面考察智能手机所遭遇的各种情况，不能很好地反映用户在真实环境中的使用体验。
- **这些评测会逐个分析智能手机的特性，如外形、屏幕、拍照、摄像、操作系统、经典应用等，有些评测还会逐项打分。但是，在真实环境下，用户往往会同时利用多个硬件特性，运行多个软件，让它们相互配合来完成任务。**例如，智能手机用户小王在餐厅就餐。在等待上菜时，她打算网购一个皮包，于是打开手机，用浏览器访问在线商城。正在浏览页面时，手机屏幕上显示一条提示：有Wi-Fi网络可用，是否连接？她选择接入，浏览器弹出新标签以显示Wi-Fi确认页面，她确认后，智能手机连上Wi-Fi网络。然后，她回到先前的页面继续浏览，这时浏览器开始使用Wi-Fi网络发送并接受数据。不久，她发现了一款很好的皮包，但没有下决心购买。为了回家之后在电脑上再详细浏览，她用邮件应用将当前的页面链接发送到自己的邮箱。在这个简单的使用情景中，用户访问了智能手机的多项功能。设计良好的手机能让用户体验尽可能平滑，而设计不良的产品则会让整个过程磕磕绊绊。只孤立地测试手机的各个功能，不考虑它们在应用场景中的交互，就不能反映出智能手机在日常生活中的表现。
- **这些评测为了表现公正客观，刻意在形式上排除个人情绪的因素。但是，用户在使用智能手机时总是带有或强或弱的情绪。**有时强烈的情感会影响他的使用方式，反之智能手机的表现也会严重影响用户的情绪。例如，智能手机用户小王受邀参与一场婚礼。在婚

礼现场，照明灯光缓缓关闭，婚礼进行曲渐渐响起，聚光灯打向大厅入口，新人步入殿堂。这时，全场欢腾，大家纷纷喝彩，并拿出手机拍照。小王也受到全场气氛的鼓舞，很开心地拿出手机拍照。设计优秀的手机能让她在黑暗中方便地启动拍照程序，快速地拍摄出高质量的照片，并轻松地分享给她的好友，这让小王在欢乐氛围中感到由衷的高兴。设计不良的手机会让小王遭遇各种问题：在黑暗的环境操作不便、拍照应用启动缓慢、拍摄的照片效果不佳等。这破坏了小王原本快乐的心情，让她感到焦急和不快。平日里可以容忍的迟缓和卡顿，在如此欢乐的氛围中却格外令人恼怒。实际上，市场上的领军产品的共同特征是能够激发出用户强烈的正面情绪，将产品“融入”其生活，让他或她不假思索地说出：我爱“我的”手机。可见，为了构建卓越的产品，测试人员需要考虑用户情感。

评测是一种简化的测试，其目标通常是在很短的时间内大致了解产品的各方面，以发布宽泛的评测结果。因此，它们拥有以上不足也许是可以接受的。不过，也有的人从实际使用的角度体验并分析产品。例如，科技作家Paul Thurrott在评测Surface Pro时，编写了一组文章[①]：总评、第一天的使用体验、在旅行中使用Surface Pro、用Surface Pro取代台式机、用Surface Pro取代超级本、对比Surface Pro和Surface RT的阅读体验等。他分析了不同场景中Surface Pro的优缺点，并讨论了如何克服产品的不足以获得更好的体验。这组文章并没有提供大量的定量分析，但是基于使用情景的定性分析更贴近用户的实际体验。

作为专业人士，测试人员更要深入考察移动应用的使用情景、功能交互和用户体验。在竞争日趋激烈的移动应用市场，只有优秀的软件才能获得用户的青睐。为此，测试人员需要深入用户情景，去全面评估软件对用户的价值。传统软件的测试人员也可以参考本节的测试方法和背后的思想，将它们移植到自己的项目，为提高用户价值作出贡献。

5.4.4　实施漫游测试

由以上讨论可知，漫游测试是一个测试方法家族，不同的测试专家从各自的实践出发，提出了不同的漫游测试方法。测试人员学习漫游测试，不但要了解具体的实施方法，还要体会漫游方法针对的问题，唯有如此才能选择合适的方法来测试产品。在选择漫游方法时，他可以考虑如下策略。

- **在项目早期，选择广泛漫游的策略**。使用功能漫游、业务漫游、价值漫游、地标漫游等方法，学习产品并发现明显的错误。测试的重心不是深入模块或功能，不是挖掘隐藏很深的缺陷，而是理解产品的远景和价值，建立功能之间的联系，获得产品的整体模型。
- **随着项目发展，根据项目风险和已发现的缺陷，选择一些深入漫游的测试方法**。使用极限值漫游、复杂性漫游、后巷漫游、阵雨漫游等方法，针对特定目标实施细致的测试。

① http://winsupersite.com/windows-8/going-pro。

通常，之前执行的功能漫游、风险识别漫游等“探索型、学习型”的方法可以为深入测试提供测试想法和风险列表。

- **根据想要了解的信息，选择合适的漫游方法**。在项目的任何时候，如果测试人员想要获得特定的信息，他都可以选择合适的漫游方法进行探索。例如，他希望了解软件是如何处理数据的，他可以选择数据漫游、数据采样漫游、快递漫游等方法来探索软件的数据处理。此时，漫游测试是学习驱动的。
- **根据想要缓解的风险，选择合适的漫游方法**。在项目的任何时候，如果测试人员想要探测软件是否具有特定类型的缺陷，他都可以选择合适的漫游方法进行攻击。例如，他感觉软件在并发计算上并不稳定，于是他选择阵雨漫游、测一送一漫游等方法来强力探测。此时，漫游测试是风险驱动的。

因为一个具体的漫游方法只探索了软件的一个方面，测试人员应该选择多种漫游方法来实施多角度的调查，并随着产品的发展引入有针对性的漫游方法。他可以通过漫游测试去掌握更多的知识，识别出可能的风险，然后安排新的漫游测试去检查风险是否暴露为缺陷。本质上，运用漫游测试是一个动态的学习和探索的过程。在具体实施漫游测试时，测试人员可以考虑如下策略。

- **利用测程实施漫游测试**。即便测试人员不使用基于测程的测试管理（详见3.2.13节和4.2.2节），他也可以拿出一个小时的时间，作为一个测程，实施主题明确的、时间固定的、专心致志的漫游测试。为此，测程的主题应该点出具体的漫游方法，例如“使用功能漫游来探索软件的功能，以建立描述功能联系的功能列表”、“使用风险识别漫游来探索缺陷数最多的3个模块，以获得一份指导未来测试的风险列表”、“使用阵雨漫游和测一送一漫游来测试模块A，以发现并发计算的缺陷”。
- **利用测试发现，实施新的漫游**。在从事某项测试活动时，测试人员发现了一些奇怪的现象或产生了一些新的测试想法，为了一探究竟，他可以选择合适的漫游方法立即展开测试。该策略常被称为“顺路游”，即暂时偏离原测试目标，对其他目标进行测试[Wittaker09]。该策略可以为测试引入变化，并捕捉到原测试策略没有考虑到的缺陷。如果“顺路游”可能产生复杂的场景，需要大量的时间，会阻碍既定测试目标的完成，那么测试人员应该记录该测试思路，并回到原先的测试任务上。这就像在地图上增加了一个地标，测试人员稍后会仔细探索该地区。

此外，测试人员还可以根据项目特征，提出自己的漫游方法，以实施更有效率的漫游。比较正式的方法是参考5.2节讨论的启发式方法创建流程，通过分析问题、提炼模式、提出方案、试验方案来构建自己的方法。比较简便的做法是修改现有的漫游方法，获得针对当前产品的漫游。例如，测试人员测试一款生成报表的产品，他可以将收藏家漫游修改为“报表收集漫游”。该漫游要求收集产品可以生成的各种类型的报表，包括空报表、最长的报表、跨年报表、跨月报表、不包含数据图的报表、包含最多数据图的报表等。执行该漫游，测试人员不但测试了报表输出，还对报表的输入数据、报表的生成条件、报表的排版等方面有了更深的认识。

5.5　快速测试

快速测试是一组基于风险的测试方法，其特征是针对某种常见的软件缺陷，实施专项攻击，通常不需要大量的测试时间和产品相关知识[Kaner11]。以下是两个例子。

- 测试人员在测试数值输入时，一般会使用极限值测试法，即填入最大值、大于最大值的值、最小值、小于最小值的值、零值等极限值。这是因为许多程序不能妥善处理极限值或极限值的组合，会发生计算错误。
- 测试人员在测试输入字段时，常采用非法值测试法。他会在字段中填入超长的字符串或空字符串、在要求输入数值的字段中填入非数值字符串、在要求输入日期的字段中填入无意义的字符串。这些攻击都检查软件能否正确检查并过滤外部输入，而不是让恶意的输入破坏其状态。

在上述案例中，测试人员只需要了解产品的基本情况，就能够实施许多测试，并可以在很短的时间内发现一批错误。

许多测试人员用测试想法列表（如3.2.8节介绍的“测试启发式方法备忘单”[QualityTree06]）或检查列表（如3.2.11节介绍的“You Are Note Done Yet: Checklist”[Hunter10]）来记录一组相关的快速测试想法。在许多情况下，这些列表是比详细测试用例更合适的测试文档。以下是它们的主要优点。

- 测试想法列表和检测列表只记录了基本的测试手段和少量具体案例，因此篇幅短小，更容易编写和维护。
- 在实际测试时，测试人员根据列表可以设计出大量具体的测试用例，即列表指明了测试策略，而具体战术则由测试人员来决定。这有助于发挥测试人员的能动性，激发出更多样化的测试设计，从而避免了刻板测试用例只测试特定操作的局限性。
- 许多具体的测试用例不会发现错误，因此无需记录在案。少量测试用例发现了重要的缺陷或值得调查的问题，测试人员可以将它们的核心想法纳入列表。这意味着测试人员通过实际测试来逐步完善列表，且无需在文档更新上花费大量时间。
- 测试想法列表和检查列表可以复用于多个项目，使得投入在它们身上的测试资源有更好的回报。

快速测试的优势在于用较少的时间快速发现特定类型的错误。如果测试人员发现他经常遇到一些相似的缺陷，他应该考虑将相应的快速测试方法加入测试计划。如果现有的快速测试方法不能发现此类缺陷（这是常见情况，毕竟每个项目都有其特殊性），测试人员可以设计一个快速测试方法，来实施专项攻击。他可以参考5.2节介绍的启发式方法建立流程，通过“分析→设计→实验→再分析”的迭代来获得有效的测试方法。

因为快速测试仅针对一些常见的软件缺陷，只依赖快速测试会错过其他缺陷。而且，快速测试没有充分利用产品的知识，因此不能发现许多领域相关的缺陷。可见，测试人员应该将快速测试与其他测试方法混合使用。例如，他用功能漫游建立产品的功能模型，用价值漫游理解软件如何服务于用户，即通过基于覆盖的测试方法建立对软件的整体认识。在此基础上，他选择一组快速测试方法对一些功能和情景进行攻击，以快速发现一批典型缺陷。根据所发现的缺陷，他可以安排进一步的测试，以探索软件或实施后继攻击。

5.5.1 James Bach的方法

测试专家James Bach和Michael Bolton在他们主持的测试培训“快速软件测试”中，介绍了一批他们常用的快速测试方法[Bach11]。其中一些方法属于漫游测试（如变量漫游、文件漫游、文档漫游等），已经在5.4.1节介绍。本节将介绍余下的快速测试方法，详情请参考表5-4。

表5-4 James Bach和Michael Bolton的快速测试方法

方法名	所针对风险	测试手段
快乐路径	软件在典型用户情景中失败	测试人员测试产品最简单、最直观、最典型的情景，完成一项或多项用户任务。在此过程中，检查其表现是否符合用户和产品团队对它的期望，而不会让用户感到疑惑、恼怒、挫折等负面情绪
中断	恰当地处理所有中断是一项有难度的开发任务，许多软件会因为中断发生故障	测试人员启动一项软件操作，立即中断它，然后再次执行。或者，测试人员启动一项软件操作，立即暂停该操作，然后恢复执行。中断的方法通常是点击“取消”按钮、关闭对话框等。在移动平台上，测试人员可以快速地跳转到软件（或操作系统）的开始屏幕，再返回
挖墙角	软件不能正确处理一些异常情况	测试人员启动一项软件操作，然后破坏该操作所依赖的资源，例如删除它要访问的文件、关闭它将访问的网络服务、启动另一个程序去锁住它要修改的数据库表格等。软件应该妥善地处理这些异常，合理地报告所遭遇的问题，不导致严重的故障
调整	软件不能正确处理数据的变化	测试人员用一些值配置软件的参数，并在软件运行过程中，修改这些参数值，以检查软件能否接受数据的变化
狗刨	当某些操作被反复执行时，软件可能出错	测试人员将一组操作重复多次，用并发的流程、嵌套的结构去考验软件。例如，文本编辑软件支持嵌套文本框，于是测试人员就不断在文本框加入新文本框，以增加嵌套层次。当文本框嵌套层次达到上限时，操作这组文本框有可能发现隐藏的缺陷
持续使用	软件可能存在资源泄漏、误差累积等长时间运行才能暴露的缺陷	测试人员长时间测试一款软件，不关闭它。在此过程中，用一些工具观察它所使用的资源，包括CPU、内存、操作系统句柄、网络端口、数据库连接等，以发现软件在资源使用方面的缺陷。此外，他还需要注意软件的功能是否随测试时间延长而变得不正确或不稳定
功能交互	不同的功能可能由不同的程序员（或同一个程序员在不同时间）编写，它们的逻辑可能不一致	测试人员发现相互调用或共享数据的一组功能，然后用夸张的数据或操作来压迫它们，以暴露交互中存在的问题

（续）

方法名	所针对风险	测试手段
点击帮助	软件不能正确显示上下文相关帮助	测试人员漫游软件的功能，如果当前功能提供了上下文相关的帮助，测试人员调出该帮助，检查所显示的内容确实描述了当前功能，能够帮助用户完成当前操作
输入约束工具	程序员在编写输入检查代码时可能犯错，一些错误的输入数据会通过检查，从而危害到软件	测试人员发现软件的输入变量，尝试各种攻击手段让非法数据通过输入检查代码
疯狂点击	资源泄漏和软件崩溃	测试人员用鼠标疯狂地点击界面的每一个像素，并持续敲击键盘。该测试也可以由一个程序来自动完成，这样的程序通常被成为“测试猴子”。整晚运行测试猴子可以发现一些资源泄漏、软件崩溃方面的错误
猛踩测试	软件不能正确处理多个并发的输入	该测试的隐喻是用户反复猛踩键盘将暴露意想不到的软件错误。测试人员可以在每个界面都实施此类攻击，也可用测试猴子来完成长时间的测试
眨眼测试	软件有时会输出一些错误的数据，但是这些错误数据隐藏在大量的正确输出中，不容易发现	测试人员将软件输出的数据可视化，使得明显的错误值可以在瞬间被识别。例如，他用电子表格加载软件的输出数据，然后绘制散点图。如果某几个点不符合整体变化规律，远远地偏在一边，它们很可能有问题。测试人员需要对它们做深入调查
错误消息后遗症	软件的异常处理代码常常有缺陷	软件弹出错误消息表明它遇到异常情况，其内部状态可能遭到了破坏。因此，测试人员一旦发现错误消息，就要对软件实施猛烈的攻击，以发现不完善的异常处理的“后遗症”
资源饥饿	在资源紧张的情况下，软件可能犯错	测试人员逐渐限制软件可以使用的资源，例如CPU、内存、磁盘、网络带宽等，直至软件报错或退出。在此过程中，软件应该平稳地对待资源限制，并避免用户数据丢失、软件崩溃、操作系统崩溃等事故
多个实例	并发运行的多个实例会相互干扰	测试人员启动软件的多个进程，并循环操作它们，以发现由并发操作造成的访问冲突、数据破坏等问题
疯狂配置	软件不能在特定配置的平台上运行	用户会修改操作系统配置，例如一些视力减弱的老年人将系统字体设为最大，或将分辨率调为最低，以方便阅读。这些配置可能会破坏软件的界面布局，使某些功能不能被正常访问。为此，测试人员需要用一些极端的配置来测试软件，以发现由特殊配置引发的错误
廉价工具	复杂的工具有时不如一些轻便的工具简单直接	James Bach分享了他使用的工具：InCtrl5[①]（环境监视工具）、Process Monitor[②]（进程监视工具）、Application Verifier[③]（调试辅助工具）、Perfmon（Windows性能监视器）、Process Explorer[④]（进程查看工具）、Windows任务管理器、一个便宜的摄像头和一款便宜的摄影机（用于拍摄测试过程）、Microsoft Excel（用于表格制作和数据分析）、HTML验证器[⑤]等。测试人员需要持续留意有价值的工具，以丰富自己的武器库

① http://en.wikipedia.org/wiki/InCtrl5。
② http://technet.microsoft.com/es-ar/sysinternals/bb896645。
③ http://msdn.microsoft.com/en-us/library/ms220948(v=VS.90).aspx。
④ http://technet.microsoft.com/en-us/sysinternals/bb896653。
⑤ http://validator.w3.org/。

不难看出，以上测试方法与5.4.2节介绍的一些漫游测试方法很相似，例如“持续使用”和“通宵漫游”都建议长时间使用产品，“多个实例”和“测一送一”都建议同时运行被测软件的多个进程。这说明不同的测试实践者针对常见的软件错误，不约而同地提出了相似的测试策略。对于测试人员而言，重新思考已有的测试方法，将它改造为适合新情景的新方法，是掌握测试并提高自身能力的必由之路。恰如Cem Kaner等所说“你不能掌握测试，除非你重新发明它”[Kaner01]。测试人员不应该单纯地接纳测试技术，他应该分析测试技术的细节，理解其工作原理，融入自己的经验和其他技术，以获得新的方法。坚持这样的研究和发明将帮助测试人员走向精通之路。

5.5.2 Cem Kaner的方法

Cem Kaner在他主持的测试课程“黑盒软件测试”中，也介绍了一批他总结的快速测试方法[Kaner11]。表5-5总结了这批方法的特征。

表5-5 Cem Kaner的快速测试方法

测试对象	所针对风险	测试手段
用户界面	用户对软件的第一印象往往来自其界面。好的界面让人愉悦，差的界面令人乏味。如果软件在应用商店中销售，其界面很可能成为用户是否安装或购买的重要因素	测试人员漫游用户界面，发现是否有任何令人疑惑、不快、烦躁的界面设计。此外，他还可以参考受到好评的界面设计，以发现被测产品的可改进之处
边界	软件在处理数值边界时可能出错	常见的测试想法： • 发现业务的范围和边界，测试边界上的值和临近边界的值 • 测试超出范围的值 • 令数值计算产生超出范围的值 • 令一个模块输出其他模块不能接受的值，例如，界面的字段只能显示20个字符的字符串，测试人员让后台模块产生21个字符的字符串。又例如，后台数据库的字段是8位整数，测试人员让前台网页传递大于255的整数
溢出	整数和浮点数运算可能溢出，这会导致计算错误和安全漏洞	测试人员要留意软件的数值计算。如果觉得某个计算有溢出的风险，就传入极大值或极小值去触发潜在的溢出
计算和操作	数据转换和数值计算有一些典型的错误	常见的测试想法： • 输入错误的数据以测试数据转换，例如，软件期望用户输入整数，测试人员就输入浮点数、无效字符串、空字符串 • 除零 • 除很接近零的极小数值，这会产生很大的计算结果，有可能导致数值越界或溢出 • 将高精度的数值强制赋给低精度的数值，这可能导致精度丢失，甚至计算错误 • 浮点数运算很可能产生误差，一些小的误差会累积出严重的偏差，因此，测试人员要关注涉及多个变量或步骤的数值计算，仔细检查其结果的精度

（续）

测试对象	所针对风险	测试手段
初始状态	一段程序在使用一个变量时，可能对该变量的状态有不正确的假设，这会导致软件故障	变量通常有如下5个状态，测试人员要检查程序的对变量状态的假设是否总是正确的： ● 未初始化 ● 已初始化 ● 默认值 ● 已经被赋予当前计算需要的值 ● 携带有其他计算的结果
被修改的值	软件可能不能侦测到它所依赖的事物发生了变化，这会导致错误	为一个变量设置值，让一些功能依赖于该变量，然后修改该变量的值，并检查相应功能是否依然正确。例如，Word允许在文档中插入"域"，以显示文件的保存时间。测试人员在文档中插入文档保存时间的域，然后更新操作系统的时区，那么文档再次保存后，时间字符串应该得到正确地更新。再例如，用图片应用浏览手机上的图片，切换到操作系统界面下删除一些正在显示的图片，再返回图片应用。此时，图片应用会如何显示那些被删除的图片
控制流	控制流的意外跳转可能暴露隐藏的问题	测试人员要想办法打乱软件的控制流，例如触发异常、输入异常数据、造成环境失败等
序列	有些软件错误会在漫长的操作序列中暴露	考虑反复执行对以下操作： ● 产生错误消息的操作 ● 令任务被迫中断的操作 ● 会执行递归计算的操作 此外，测试人员可以编写测试程序，用随机的顺序调用已有的自动化测试用例，实施长序列的测试
消息	软件与外界的通讯可能中断	如果软件与网络服务、数据库、其他软件有任何形式的交互，测试人员要破坏它们之间的通讯。他可以编写程序来监听并篡改消息，或强制被依赖的软件返回错误的消息，以测试软件如何处理破损的消息
时序和竞态条件	软件可能对操作的时序有不正确地假设	测试异步调用时，测试人员让软件先后启动A和B两个计算，并让B先完成计算。有些软件不能处理后提交的计算先返回的情况，会发生故障。此外，测试人员可以同时启动多个操作去访问共享的数据或功能，以检查并发计算是否造成数据损坏
干扰测试	软件在执行一段关键任务时，会因为某些干扰而失败。有些失败是合理的（但不应该造成故障和数据丢失），有些失败则是可以避免的	如果一个任务会持续一段时间（例如，软件显示了进度条或等待提示符），测试人员可以在其执行过程中产生一些干扰。 ● 移除该任务依赖的资源。 ● 取消该任务。 ● 暂停该任务，然后继续。 ● 对于移动应用，先切换到操作系统的开始屏幕或其他应用，再返回。 ● 启动其他程序与被测软件抢夺资源
错误处理	软件的错误处理代码需要应对意外且棘手的情况，有较高的编写难度，因此容易出错	测试人员尝试着触发软件的错误消息，然后反复执行导致错误消息的操作，以检查错误处理代码是否产生了资源泄漏等问题。然后，他强力测试软件，以检查错误处理代码是否妥当地将软件恢复到正确的状态

（续）

测试对象	所针对风险	测试手段
失败处理	软件的失败恢复可能包含错误，使得损失加重	测试人员要检查失败处理代码的正确性。例如，许多文本编辑软件提供了自动恢复功能，当软件崩溃重启后，它可以显示用户最近的编辑成果，避免或降低了用户数据的丢失。测试人员要测试自动恢复确实可以恢复文档中的各种数据
文件系统	文件系统的异常情况会影响软件的运行	常见的测试想法： ● 用调试工具（如Application Verifier）令操作系统的文件读写API返回错误，这可以模拟出磁盘损坏等异常情况 ● 删除正在被使用的文件 ● 删除即将被使用的文件 ● 锁住即将被使用的文件 ● 窃取软件持有的文件句柄（杀毒软件在扫描硬盘文件时会窃取文件句柄） ● 修改一个需要更高权限才能修改的文件 ● 令文件名或文件路径的长度超过操作系统支持的最大路径长度 ● 令文件名包含操作系统禁止的字符 ● 将文件名修改对操作系统有特殊含义的单词，例如NUL在Windows系统上是一个特殊的设备名，不能是文件名 ● 读写尺寸巨大的文件 ● 将文件写入只读的磁盘 ● 将文件写入容量即将耗尽的磁盘 ● 将文件写入网络磁盘，在写入过程中，断开网络 ● 将文件写入移动硬盘，在写入过程中，拔出移动硬盘
负载和压力	在高负载的压力下，软件可能出错	测试人员想办法提高软件的工作负载，并制造资源短缺的情况，以测试软件在性能压力下的行为
配置	软件可能存在兼容性缺陷	测试人员改变操作系统的配置，如日期格式、时间格式、货币格式、时区、默认语言等，以检查软件在不同系统配置下的行为
多变量关系	多个变量的交互可能诱发软件错误	常见的测试想法： ● 用两个有效的值组成非法的取值组合，例如“2月30日” ● 用相似的操作来处理不同的对象，例如在Word中剪贴一组对象（文字、文本框、图片、形状、艺术字、域、表格等）

5.5.3　James Whittaker的方法

James Whittaker的测试名著*How to Break Software*（HTBS），以失败模型为基础，介绍了一批测试方法[Whittaker01]。这些方法大多属于快速测试，能够针对特定类型的缺陷实施便捷的攻击。

James的缺陷模型是一个讨论和理解软件缺陷的框架，他使用该框架来组织典型的软件失败和相应的发现手段。如图5-9所示，失败模式主要关注点是环境和能力。

- 环境是软件依赖或影响的外部对象，典型的例子是人类用户、文件系统、操作系统和与之协作的软件。HTBS写作于2001年，所以它主要考虑了单机软件的环境。现今，网络应用和移动应用已经成为主流软件，测试人员需要考虑的外部对象还应该包括硬件设备（如陀螺仪、加速感应器、光线感应器、GPS等）和网络服务。
- 能力是软件功能达成用户价值的手段，该定义与Google ACC的部件通过能力实现属性的理念是一致的（Google ACC的讨论参见3.2.2节）。HTBS针对能力实施测试，主要测试切入点包括软件的输入、输出、数据、计算和集成。

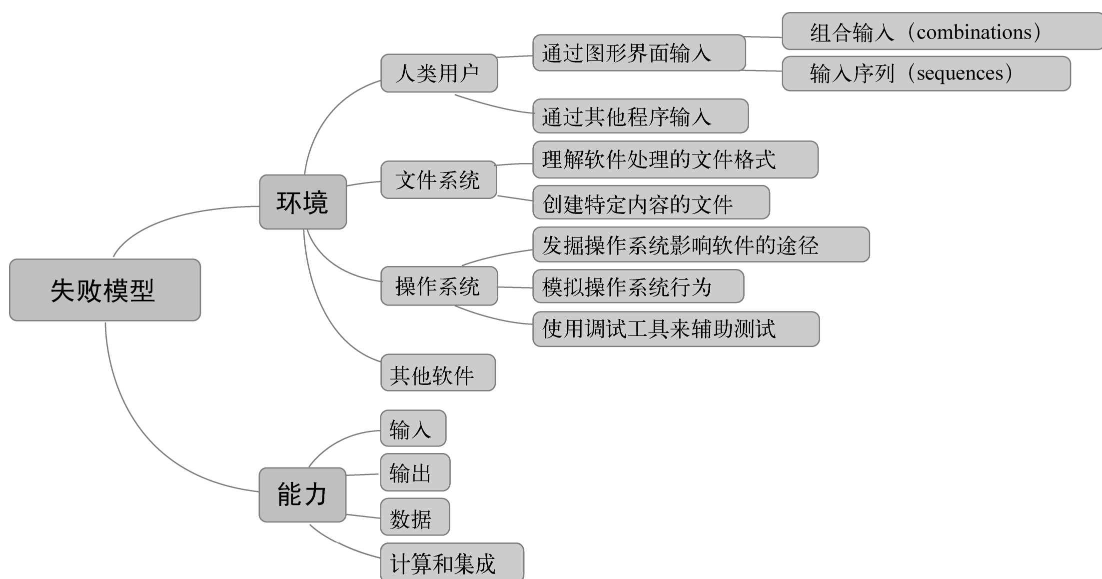


图5-9　失败模型关注环境和能力

James Whittaker建议测试人员先实施漫游测试，以识别软件的界面（HTSM→产品元素→界面）和相应的外部对象，然后列出外部对象在该界面上可以执行的输入和观察到的输出（4.2.2节讨论了输入与输出模型）。此外，测试人员还需要通过功能漫游识别出软件的主要功能，列出这批功能所持有的数据和所执行的计算，并理解功能之间的依赖和协作关系。

在掌握了软件的基本情况后，测试人员可以根据错误模型对软件进行测试。他专注于软件的一个界面、一项能力和一个可能的失败，实施快速的攻击。当攻击完成后，他转向下一个目标再进行攻击，即通过一场场小规模的战斗来赢得一场战役（赢得整场战争还需要使用更多的测试方法）。表5-6是HTBS所建议的攻击手段。

表5-6　HTBS的攻击手段

攻击手段	所针对的软件失败	如何实施攻击
通过用户界面攻击软件的输入		
尝试各种输入以触发软件的所有错误消息	软件处理错误输入的常见方式有三种。 • 输入过滤：滤除错误的数据。 • 输入检查：检查输入并报告错误。 • 异常处理：当错误输入产生故障时，异常处理代码会捕获异常并加以补救。 正确地编写以上代码需要周密的思考和反复的测试，稍有不慎就可能引入缺陷	采用极限值测试、边界值测试、非法值测试、压力测试等方法来测试用户输入
强制让软件使用默认值	不同的模块由不同的程序员编写，他们可能期望不同的默认值，这种不一致性会导致错误	常见的测试想法： • 接受所有默认值，然后提交 • 使用空值 • 将默认值改为另一个值，然后再改回来 • 将默认值改为另一个值，然后改为空值
探索允许的字符集和数据类型	常见问题。 • 字符集：有些软件只支持特定编码的字符集，输入其他编码的字符会导致错误。 • 编程语言：有些软件会将用户输入作为程序来执行，这导致了许多问题，如SQL注入。 • 操作系统：操作系统会禁止创建一些特定名字的文件。一些软件在西文操作系统上不能正确显示中文字符	测试人员了解软件、编程语言、实现技术、操作系统对字符集和数据的期望与限制，然后使用对它们而言具有特殊含义的字符串或数据进行攻击
令输入缓冲区溢出	软件没有考虑输入值超长、超大的情况，导致数值计算溢出或缓冲区溢出	常见的测试想法： • 输入超长的字符串，字符串最好类似于“壹贰叁肆伍陆柒捌玖拾1234567890壹……”，这有助于定位导致错误的长度 • 输入极大值（针对加法和乘法计算）、负的极大值（针对减法计算）、很接近零的值（针对除法计算）等
测试一组相关变量的取值组合	当多个程序员共同开发一段代码时，他们的代码所产生的数值可能不能正常协作。此外，程序员有时会用复杂的嵌套if语句来检查多个变量，这些代码不容易编写，更容易在维护时出错	测试人员通过调查软件需求和实现，以识别真正相关的变量，它们通常属于同一个数据结构或参与到同一项计算中。然后，根据软件使用它们的方式设计测试用例。此外，也可以使用4.1节介绍的组合测试工具来产生测试用例
重复输入多次	软件可能不了解它在空间和时间上的限制，当重复操作耗尽其资源时，它会失败	测试人员反复执行一些消耗较多资源的操作，或反复执行一些产生大量数据的操作。例如，在测试在线商城时，测试人员持续向购物车中添加货品，创建出一个包含许多商品的“超级订单”。该订单可能导致提交失败，或令未来的订单处理遇到困难

（续）

攻击手段	所针对的软件失败	如何实施攻击
通过用户界面攻击软件的输出		
强制软件对相同的输入产生不同的结果	软件的输出不仅取决于输入，还取决于其内部状态、所依赖的外部数据和程序。如果程序员没有妥善地考虑这些因素，软件很可能能在用户输入、自身状态、外部依赖的共同作用下失败	测试人员需要咨询领域专家、产品经理和程序员，以了解软件的业务规则和代码实现。然后，通过测试来建立软件的模型，如4.2.2节讨论的输入与输入模型、4.2.5节讨论的状态机模型等，以掌握软件如何处理外部因素和内部状态。在此过程中，他用该攻击方法来探测软件的缺陷
强制软件产生不正确的输出	该攻击要求测试人员"逆向思考"，从输出推测导致错误的输入。许多测试人员觉得这种思考方式不自然。同样，程序员也不习惯于这样思考，所以他们的设计可能忽略了一些特殊情况，而埋下了错误	在战术上，测试人员可以采用与上一条攻击相似的测试流程，通过迭代的学习、建模和攻击来寻找缺陷。不同之处在于本攻击更强调暴露输出错误。例如，测试日期选择面板时，测试人员先选择闰年的2月29日，然后点击"下一年"。那么日期选择面板会显示非闰年的2月29日吗
强制一个输出的属性发生变化	有些输出包含多个属性，例如PowerPoint在导出图片时就需要考虑原对象的色彩、对比度、亮度、特效、边框等属性。协同工作的代码（调整对象属性的代码、在屏幕上绘制对象的代码和导出图片的代码）可能存在不一致性	测试人员令输出包含各种属性，然后逐一调整属性，并检查新的输出。在调整属性时，要重点考虑极限值、非法值和数据性质的变化。例如，属性的效果从无到有（比如其数值从0变化到1）或从有到无（比如其数值从1变化到0）很可能会对输出结果产生重要影响
强制屏幕重新 绘制	软件的图形界面需要正确地在屏幕上绘制对象和动画，这是有挑战性的编程任务。虽然使用成熟的程序库可以避免许多典型的问题，但是软件仍可能在某些情景中失败	测试人员增加、删除、移动屏幕上对象，调整软件的界面布局，改变其窗口大小和位置，在此过程中密切关注软件有没有正确地、及时地重新绘制。屏幕上不应该出现闪烁、黑影、对象残留等不正常的情况
通过用户界面探索存储的数据		
针对一组变化的初始状态进行输入	软件在检查用户输入时可能没有考虑自身的状态，而放过了一些导致故障的输入	该攻击与"强制软件对相同的输入产生不同的结果"相似，可采用相似的测试流程。不同之处在于，之前的攻击强调细致地探索软件，通过设置软件状态来产生特定的输出。本攻击强调设置软件状态来让用户输入产生失败
强制一个数据结构容纳过多或过少的数据	固定长度的数据结构（如数组）只能容纳特定数目的对象。动态数据结构能够申请内存去容纳更多的对象，并在所容纳的对象减少时归还内存，这带来了更复杂的代码。测试人员需要检查这些数据结构可以正确处理数据的变化	常见的测试想法： • 向一个数据结构中连续添加数据，直至软件提示容器已满或暴露出错误 • 取出一个数据结构中的数据，直至软件提示容器已空或暴露出错误
发现可以修改数据的所有可行方法	如果软件提供了多种方法修改其内部数据，程序员需要对每种方法实施正确且一致的检查。在软件开发和维护过程中，该一致性可能被破坏，以至于某些方法缺少一些重要的检查	测试人员探索软件，了解软件对数据的约束，然后尝试各种方法去绕过数据约束。例如，制表功能只允许插入256行的表格，测试人员可以先插入256行的表格，然后通过"插入行"功能来突破限制

（续）

攻击手段	所针对的软件失败	如何实施攻击
通过用户界面探索计算和功能集成		
尝试非法的操作数和操作符的组合	“除零”错误是由除法运算符和操作数0共同作用引发的异常。程序员可能检查了此类典型的非法组合，但可能漏过了其他非法组合	测试人员探索软件，了解软件所使用的计算，针对它们的特点实施攻击
强制一个函数递归调用自己	编写正确的递归函数和嵌套数据结构需要一定的技巧，其代码可能包含隐蔽的错误。而且，逻辑正确的递归会因为一些现实的限制而导致错误，例如大量的函数递归调用将耗尽调用栈的内存，从而导致栈溢出异常	测试人员漫游软件，列出可能包含递归的功能，然后想办法让它无止境地递归下去。例如，测试文档编辑软件时，在图文框中持续插入嵌套图文框，使嵌套层次持续增长直至出错。又例如，在正文中插入脚注，然后在脚注中插入脚注
强制计算产生过大的结果或过小的结果	过大或过小的结构都可能造成溢出错误。此外，即便过大或过小的结果是正确的，当这些结果传递给其他功能时，那些功能可能会遭遇失败	该攻击与“令输入缓冲区溢出”相似，基本测试想法是输入极限值和特殊值，并通过连续操作令数值趋向极大值或极小值
发现不能正确分享数据或彼此交互的功能	共享数据或彼此协作的功能需要遵守一组约定。当不同的人编写这些模块时，他们可能对约定有不同的理解，这会导致不一致的实现和功能集成的错误。此外，代码修改也容易破坏实现的一致性	基本的测试想法是用一个功能去影响另一个功能的输入、输出、数据和计算。由于软件的功能很多，测试人员需要聚焦于用户价值，像真实用户那样使用软件。在此过程中，通过仔细地观察和发散地思考，识别相互关联的功能，并进行攻击
通过文件系统实施基于介质的攻击[①]		
耗尽文件系统的能力	程序员可能从没有思考过存储空间耗尽等情况，因此软件不能正确处理该异常。此外，单元测试较难触发此类情况，因此开发者测试可能漏掉了该问题	测试人员探索软件，识别出读写文件系统的操作，然后逐一测试这些操作。在具体操作之前，他需要耗尽文件系统的资源，以制造极限的情况。典型的情况包括以下几种： • 磁盘空间耗尽 • 磁盘被写保护 • 驱动器中没有盘片 • 读卡器中没有存储卡 • 访问拒绝
强迫介质忙碌或不可访问	软件不能很好地处理此类极端情况	常见的测试想法： • 使用工具快速读写磁盘，令磁盘忙碌 • 使用网络工具限制计算机带宽，降低软件访问网络磁盘的吞吐量 • 使用工具耗尽操作系统的文件句柄
介质损坏	软件不能很好地处理此类极端情况	测试人员可以使用受损的磁盘和光盘来测试软件，也可以使用工具来模拟出文件系统的错误

① 在本类攻击中，因为有些情景不容易手工创建，James Whittaker使用一款工具（Canned Heat）来模拟出文件系统的异常情况。不幸的是，该工具的开发与维护已经停止了。目前，在Windows平台上拥有相似功能的软件是微软的Application Verifier，它功能强大，但入门难度较高。不过，测试人员可以编写一些小程序来模拟出类似的情况。

（续）

攻击手段	所针对的软件失败	如何实施攻击
通过文件系统实施基于文件的攻击		
让软件去产生非法的文件名	操作系统对文件命名有一组规定，软件应该对用户提供的软件名实施相同的约束。如果它不加检查就去产生非法的文件名，就会遭遇失败	测试人员需要学习操作系统对文件名的约束，然后构造非法文件名去攻击软件
改变文件访问权限	软件在访问文件时可能遭遇多种异常。如果程序员没有思考过相关情况，软件将不能正确地处理这些异常	测试人员探索软件，识别出软件会读写的文件。在软件访问该文件之前，修改其访问权限。常见的测试想法包括以下几种： ● 将文件属性修改为只读 ● 将文件属性修改为可读写 ● 将文件的所有者修改为其他用户或用户组 ● 用其他软件打开该并锁住该文件
修改或破坏文件的内容	软件应该检查从文件中读取的值，并报告遇到的错误。不正确的文件解析和检查将导致软件故障。此外，黑客也会用文件攻击软件：软件读取一个恶意文件，在解析的过程中出错，然后被引导去执行恶意代码	常见的测试想法： ● 测试人员以一个正确的文件为起点，逐个修改其字段，以测试软件对它们实施了正确的检查 ● 测试人员破坏文件的结构，以测试软件能否正确地容忍错误或报告文件异常 ● 测试人员利用模糊测试工具产生大量的破损文件让软件读取，以暴露隐藏的缺陷
通过软件或操作系统进行攻击		
注入错误	软件很可能不能妥当处理操作系统或被依赖的软件产生的错误	测试人员制造一系列操作系统或被依赖软件的错误，以检查软件能否处理。常见的测试想法包括以下几种： ● 制造或模拟内存错误 ● 制造或模拟网络错误 ● 制造或模拟被依赖软件的异常

由以上介绍可知，HTBS的内容是一组典型的软件失败情景和相应的攻击手段。James Whittaker用环境和能力为框架，将它们有条理地组织起来，以便于学习、交流和应用。在测试工作中，测试人员应该收集典型的缺陷，分析它的原因和症状，并设计相应的测试方法或启发式指导词。一段时间后，他可以用一个框架来组织这些方法和指导词，并将成果分享给测试小组的其他成员。通过有针对性的测试设计和相互学习，测试小组可以更有效地发现产品的问题。

5.6　情景测试

情景是一个连贯一致的故事，描述了特定用户如何使用软件。情景测试将一个或一组情景作为测试方案，通过运行软件来排演这些故事，从而评估软件的行为[Kaner11]。

情景测试的主旨是从实际用户的角度考察软件在现实场景中的表现。软件产品是一项解决方

案，它帮助用户完成困难的任务，并达成业务目标。其解决问题的能力来自于一组相互支持的部件和功能。为了检查软件的能力，情景测试通过较长的业务流程来测试软件的多个功能，从而整体评估软件对用户的价值。因为情景测试不局限于特定的功能，并站在用户的立场更全面地考察软件，所以它可以发现一些整体设计上的缺陷。此外，情景测试覆盖了多个功能，因此有可能暴露出一些复杂交互导致的错误。

5.6.1 基本方法

如果缺少测试主题的指引，长流程的情景测试可能会偏离方向，变成漫无目的地游览。为了有效利用时间，明确测试方向，测试人员需要构思一个（或一组）用户使用软件的故事，然后围绕故事组织测试活动。以下是一个案例。

Microsoft Lync是一款企业级的统一通讯平台[①]，支持多种形式的即时通讯和在线会议。为了测试如图5-10所示的Lync会议功能，测试人员编写了如下故事，作为情景测试的大纲。

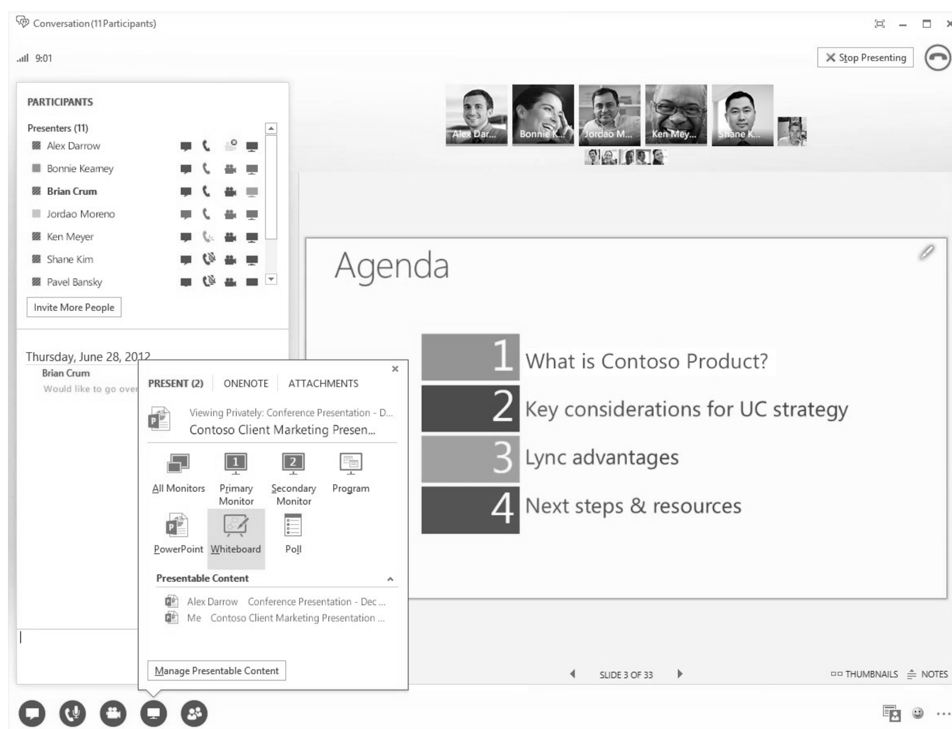


图5-10 使用Lync 2013进行视频会议

① http://lync.microsoft.com/。



Brian Crum是Contoso公司的一位经理。周一，他打算组织一个在周五举行的视频会议，邀请本公司在其他地区的同事参与，共同讨论下一季度的产品营销策略。他在Outlook创建了一个Lync会议的日程安排（包含会议议程、参与会议的拨入号码、启动Lync的链接等），发送给8个同事。周三，他用Outlook更新了会议安排，添加了更详细的会议议程，并多邀请了两个同事参加会议。周五，他在会议前5分钟打开笔记本电脑，点击启动Lync的链接，登录到Lync会议中。他检查了摄像头、麦克风和耳机，一切正常。不久，一些同事陆续进入会议，Brian通过语音通讯向他们问好，并确认他们可以看到他分享的PowerPoint幻灯片和摄像头视频流。等到所有与会者到齐，他开始会议，讲解下个季度的营销策略，并在Lync中播放相应的PowerPoint幻灯片。在此过程中，同事们通过即时消息或语音通讯提出一些问题，他一一给予回答。当回答一个复杂问题时，他调出Lync会议的白板，用电子笔透过触摸屏在白板上绘制草图，进行详细解释。然后，他切换回PowerPoint幻灯片，继续讲解。讲解完毕后，作为会议组织者的Brian将演示权赋予同事Alex，让他成为演示者。Alex用Lync展示了他的第二个屏幕，开始演示产品的新特性。与此同时，Brian在Lync会议中附上一些报表，以便与会者下载参考。等Alex结束演示，Brian重新主持会议，感谢同事Ken Meyer进行了会议记录，并提示大家参考共享的OneNote会议纪要。在获得大家认可后，他结束了Lync会议。Brian对Lync（及其服务器）在会议中的表现感到满意，愿意下次继续使用Lync来召开视频会议。

这是一个很复杂的故事，通过举办多人参与的视频会议，测试了产品家族的一组功能。

- 在Outlook中创建并更新Lync会议的日程安排，测试了Outlook与Lync的集成。
- 使用日程安排的链接启动Lync，测试了Lync会议的启动。
- 在多人会议中，通过视频、语音、文字等进行交流，这可能暴露一些点对点通讯所不能发现的缺陷。
- 使用Lync播放PowerPoint幻灯片，测试了Lync对内嵌多媒体的支持。
- 使用Lync白板绘制草图，测试了Lync对触摸屏和电子墨水的支持。
- 测试了Lync中PowerPoint与白板的来回切换。
- 让他人演示程序，测试了会议控制权和视频画面的切换。
- 使用Lync共享第二个桌面，测试了Lync对多显示器的支持。
- 组织者上传文件让与会者下载，测试了Lync对附件上传和下载的支持。
- 使用OneNote记录会议内容，测试了Lync与OneNote的集成。

在获得该故事之后，测试人员要做一些测试准备。首先，他邀请测试同事参与情景测试，向他们讲解故事和测试要点，并安排每位测试人员扮演一个或两个与会者。然后，他准备11台计算机，并搭建测试环境。这11台计算机的软硬件配置要尽可能的差异化，例如：有些计算机拥有高性能处理器和大内存，有些计算机只拥有过时的处理器和小内存；有些台式机拥有外置的高清摄

像头、高档耳机和麦克风，有些笔记本电脑只拥有分辨率很低的内置摄像头、效果一般的内置麦克风与音响；有些计算机用电缆接入高速网络，有些计算机接入无线网络（Wi-Fi网络、3G网络等）。这样做的目的是尽可能模拟出各类用户接入Lync会议时实际使用的设备和网络。

在实际测试时，每位测试人员在独立的房间中扮演各自的角色，以模拟出远程会议的特点。他们执行相应的操作，并密切注意任何异常情况，如视频的闪烁、幻灯片播放的停顿、音频的爆音等。如果一位测试人员发现问题，他会用Lync通知测试小组暂停测试，以便他收集本机或服务端的日志、内存转储、硬件配置等信息，从而报告问题的技术细节。此外，测试小组还可以根据他遇到的问题，构思出几个小的故事，再做一些额外的调查，以获得更多的信息。在一个长流程的情景测试中，暂时偏离测试方案，做一些短暂的攻击，是常见的测试策略。这有助于抓住测试中偶然发现的信息，发掘出有价值的信息或隐藏的缺陷。一段时间后，测试组织者应该提示测试小组回到原先的故事继续测试。如果测试延续的时间很长，测试小组可以中途休息一会，一起吃点东西喝点饮料，并讨论接下来的测试重点。在测试结束后，每位测试人员提交所发现的缺陷，测试组织者编写测试报告，概述测试过程和覆盖范围，并提出值得进一步调查的问题和风险。

以上案例是一个多人参与的情景测试，其基本要点和步骤也适用于单人执行的情景测试。

- 编写情景测试的故事，像用户一样复杂地使用产品。
- 准备测试环境，尽可能模拟出用户的使用环境。
- 在故事指引下，积极地测试，并根据测试发现，即席设计并执行一些测试。故事是地图，如何具体地探索需要测试人员来动态导航。
- 测试人员提交缺陷报告，并记录需要再调查的区域和风险。

在整个测试准备和执行过程中，测试人员就像是集编剧、导演和主演于一身的电影工作者。首先，他像编剧一样书写测试剧本，安排情节大纲，构思出引人入胜的故事。然后，他成为导演，筹措测试需要的资源，并总控测试进程。在具体测试时，他像演员一样去推敲剧中人物的特点和风格，既参考现有剧本，又即兴发挥，在故事的留白处尽情演绎。有时，他会回到导演的角度，临时加一场戏，以更生动地展开故事，或者反思全局，以动态调整故事的发展方向。

Cem Kaner对于情景测试有长期且深入的研究。他提出，一个好的情景测试通常会具有如下特征[Kaner11]。

- **基于连贯一致的故事**。情景测试描述了用户如何使用软件，并关注他们的目标和情感。为此，测试人员需要思考并确定用户的特点、他们的目标、他们使用软件方式、他们在故事中的情绪等要素。
- **故事是可信的**。当测试人员将故事描述给项目关系人，他们会认为故事很可能发生在用户环境中。这并不意味着故事只能覆盖产品的快乐路径，而是要求测试人员深刻理解用户的任务和环境，构思出有洞察力的故事。

- **故事是有推动力的**。当测试人员报告故事所发现的缺陷，项目关系人会认为修复该缺陷是值得的。为此，测试人员可以用一些启发式问题来帮助故事设计：谁会重视这个故事描述的情景？他们为什么会重视这个情景？如果该情景失败，它会影响到谁？失败造成的最严重后果是什么？如果该情景失败，用户有没有替代方案？如果替代方案与被测产品无关，那么用户为什么要使用该产品？
- **故事引入了复杂性**。测试人员在复杂的环境中以复杂的方式使用软件。故事的复杂性根源于现实情景的复杂性，采访用户了解他们的实际遭遇、使用他们的真实数据、模拟他们的操作环境，都有助于制定复杂且可信的故事。
- **测试结果易于评估**。测试人员和项目关系人能够轻松地判断测试结果并识别潜在的问题，这对于复杂的故事尤为重要。使用5.3节介绍的基于一致性的测试先知和基于约束规则的测试先知，能够较快速地判定测试结果。

Cem Kaner还指出情景测试能够超越简单的测试，深入地调查软件是否能交付它应该交付的价值[Kaner11]。有一些测试方法能够发现用户在实际使用的头几天会发现的缺陷。例如，超模漫游专注于用户界面的测试，旨在快速发现控件、文字、动画等在视觉呈现上的错误。如果此类错误被研发团队忽视，它们很快会被用户察觉，而且软件的用户越多，发现时间越短。好的情景测试则可以发现资深用户才会遇到的问题。这批用户长时间的使用产品，积累了几个月甚至几年的经验，会利用它完成重大的工作。情景测试有助于“浓缩”使用时间和用户经验，在较短的时间内模拟出资深用户的高级用法、困难使命和复杂语境，发现软件的深层次问题。

因为情景测试会覆盖多个功能，它要求被测试的功能比较稳定，较少出现阻碍故事推进的严重错误。因此，测试人员需要在功能级别做足够的测试，使单独的功能足够稳定，然后做一些集成级别的测试，使功能协作足够流畅，之后才进行长流程的情景测试。这说明测试人员需要利用多种测试技术，在功能、集成、系统等层面进行有针对性的测试，才能逐步完成对软件产品的严密测试。

5.6.2　设计用户角色

情景测试的核心是情景，即一个连贯一致的故事。一个好的情景将帮助测试执行更有效地探索产品。本节将介绍一些编写情景的方法和技巧。

情景是讲述一位（或几位）用户的故事，因此测试人员需要全面地理解用户，了解其使用软件的目的、需要完成的任务和的操作软件的特征。用户角色是一种常见的用户建模方法，能够帮助测试人员从用户的角度思考软件。开发专家Mike Cohn提出了一种构建用户职责的方法，可以有效地描绘出产品需要支持的人群，并自然衍生出用户角色[Cohn04]。基于该方法，测试人员可以按如下步骤构建用户角色。

- **利用头脑风暴，列出可能的用户角色**。测试小组（或项目团队）组建“用户角色工作小组”，进行头脑风暴会议，列出一组产品的用户角色。所谓用户角色是一个可以代表一组

用户的人物，他反应了一个用户群所共有的特征。在这个阶段，与会者的目标是记录尽可能多的用户角色，而不必在意这些用户是否重叠、所代表用户群的是否庞大、对产品成功是否重要等因素。

- **整理初始的用户角色，识别出目标和特征有重叠的用户角色**。
- **整合用户角色**。如果两个角色有重叠，可以创建一个新的角色来涵盖他们的特征。如果两个角色很相似，可以舍弃其中一个角色，让保留的角色反映其特征。此外，工作小组还应该舍弃对产品成功不太重要的角色，以便将精力集中在重要的角色上。
- **提炼用户角色，定义具体的角色特征**。所谓角色特征是将该角色与其他角色区别开的事实，常见的特征包括：使用产品的目的、使用产品的频率、对业务领域的熟悉程度、对计算机的熟悉程度、对产品的了解程度等。
- **添加角色的个人信息，如姓名、照片、脾气、处事风格等，令角色“有血有肉”**。用户使用产品时总带有个人特色和主观情绪，而好的软件应该做到善解人意。

以上用户建模的结果是一组产品需要重点支持的用户角色。例如，PowerPoint的测试人员经过分析和提炼，可能获得如下用户角色。他们覆盖了不同类型的用户群，能够传递出目标用户对软件的期望。



Alex是一位职业演说家，38岁，男性，经常在各地的大型场馆做演讲。他使用电子幻灯片来增强演说的感染力。对他而言，他本人才是演说唯一的主角，所以他的幻灯片看上去总是很简洁。实际上，他会对幻灯片的每一个细节反复推敲，以求最佳效果。他会放大文字的尺寸，让坐在远处的观众也可以轻松阅读。他会使用高分辨率的图片，以便在投影放大后图片依然清晰动人。他会要求每个页面的配色都和幻灯片整体协调一致，不过有时他会故意引入醒目的颜色来增强演说的戏剧效果。他对PowerPoint的一部分功能很了解，为了优质的幻灯片，他会深入尝试他感兴趣的功能，但是他需要花更多的时间在演讲的研究、构思和排演中。

Cameron是一位商业分析师，28岁，男性，几乎每周都利用电子幻灯片向客户讲解其研究报告的精华部分。他工作积极，动作很快，是Microsoft Word、Excel、PowerPoint和OneNote的重度用户。他会用OneNote保存他收集的各种资料、链接、素材和临时的想法，用Excel做数据分析，用Word编写研究报告，用PowerPoint展示来自OneNote、Excel、Word的图片、表格、数据图等对象。在编辑幻灯片时，他会同时启动多个应用程序，将大量的内容“搬运”到PowerPoint中，并实施必要的后续编辑。最后得到的幻灯片页数众多，内容丰富多样，文件尺寸巨大。有时，他会邀请同事共同编辑一个幻灯片文件，让更多的数据和观点得到呈现。总体上，他对PowerPoint的方方面面都很了解，能够熟练运用大部分功能 ，并拥有很强的学习能力。由于工作压力很大，他比较没有耐心，当产品反应迟钝或出现漫长的进度条时，他会显得有些焦躁。

Stephanie是一个大学生，20岁，女性，经常需要提交研究报告以获得课程成绩和学分。有时，任课的教授不但要求学生提交书面报告，还会面试他们，要求他们进行口头报告。为了提高课程成绩，Stephanie会制作电子幻灯片来帮助宣讲。她是一个聪明的学生，但是对于计算机和软件的细节不感兴趣。她的目标是尽快完成看上去不错的幻灯片，以便周末去滑雪。虽然她不会深入研究PowerPoint，但是她可不希望丑陋的幻灯片给教授留下恶劣的第一印象。如果产品能够帮助她快速制作出典雅大方的幻灯片，她会感到很满意。

根据这些用户角色，测试人员可以更流畅地设计测试故事，因为具体形象让测试人员能够快速“入戏”。当测试人员的头脑沉浸在用户角色和环境中，一些角色特有的目标、任务和情节就会自然浮想。例如，他构思出：“Alex坐飞机去遥远的城市研究，在孤单的旅程中，他断断续续地修改幻灯片，并在演讲的前夜做最后的推敲和修改。演讲时，他利用PowerPoint的演讲模式，精确地掌握演讲的进度和时间，并在演讲结束后将幻灯片发布到自己的网站。”又例如，他编写出：“老板向Cameron和其他两个同事下达指令，要求他们立即完成一份向大客户报告的幻灯片。于是，他们利用PowerPoint的共同编辑功能同时编辑一份幻灯片。大多数时候，他们工作在自己的幻灯页，但有时也会工作于同一张幻灯页。在此过程中，PowerPoint会和存储幻灯片的服务器配合，以避免潜在的修改冲突。”在构思故事时，测试人员要围绕产品期望交付的价值，用故事去检查软件确实达到了它对用户的承诺。

除了用户角色，测试人员还可以访问实际用户，去获得他们的真实故事。很多时候，面对面的交流能够提供其他任何方法不能获得的信息。通常，测试人员不是实际用户，或者只能代表广大用户群的一小部分用户。一些用户的目标、任务和行为，可能是测试人员前所未见、无法想象的。因此，如果有机会，测试人员可以利用正式或非正式的形式采访实际用户，与他们交谈或共同操作软件，以了解他们为什么使用产品、要完成哪些任务、如何操作产品、感到哪些困惑和不满、对未来产品有何种期望等。在采访过程中，测试人员可以请客户讲述几个与产品有关的印象深刻的故事。这些故事往往蕴含了用户的情感（否则不会留下深刻的影响），体现了一些深层次的目标和动机，反映了现有产品的优点或不足，值得测试人员认真参考。

除了正面角色，测试人员有时得考虑负面角色。例如，一些恶意的攻击者会在网络上散布精心构造的PowerPoint文档。当这些文档被打开时，它们会令PowerPoint执行一段恶意代码，从而控制他人的计算机或窃取私密信息。为了保护正面角色的利益，测试人员需要研究负面角色的动机、目标、能力和手段，并设计相应的测试方案。因为安全测试的特殊性和技术性，该方案需要包括一组针对特定安全风险的专项测试技术。情景测试有助于从负面角色的角度思考攻击，可以为测试方案提供有价值的想法。

在有些项目团队，产品经理会定义用户角色。测试人员可以利用这些成果，直接用于情景测试的设计。如果测试人员发现某个角色很重要，但是被产品经理遗漏了，他可以与产品经理面对面地交谈，了解背后的原因，探讨更好的角色设计。此外，有些项目团队会编写用例或用户故事

以帮助软件设计。测试人员可以秉持“拿来主义”，将这批基于情景的文案修改为针对用户角色的测试故事。有时，测试人员可以将多个用例或用户故事合并为一个大型的情景测试，以实施复杂的长流程测试。改写的基本原则仍旧是保证故事的连贯一致、可信、复杂、有推动力、易于评估。

5.6.3 情景测试与漫游测试

情景测试和漫游测试可以很好地配合与协作。在基本方法上，它们都不依赖刻板的测试脚本，而是让测试人员在游历软件的过程中发掘新信息，构思并执行新的测试用例，因此能够相互协作。在测试方向上，情景测试使用故事作为指导，专注于用户的价值、任务和体验，而漫游测试的主题则更加多样，可以是一个启发式指导词、一个隐喻、一组需要覆盖的对象、一些需要调查的问题、一批产品风险等。将它们组合使用，可以发挥各自的长处，形成互补的效果。

在执行漫游测试过程中，测试人员可以设计一些情景，通过演绎这些故事来丰富漫游测试的测试手段。以下是一些典型的例子。

- 在实施用户漫游时，测试人员需要覆盖典型的用户情景。他选择一个用户角色作为此次漫游的主角，根据他的业务情景，设计出一组故事来进行测试。在执行故事的过程中，测试人员推敲用户对软件的期望，尝试他可能使用的方法，用有挑战性的任务来检查软件对用户的支持。
- 在实施价值漫游时，测试人员根据产品远景、需求文档、市场文宣等材料确定产品期望交付的一组核心价值。针对每个核心价值，他编写几个故事，来检查软件确实达到了它的承诺。
- 在实施情绪漫游时，测试人员需要重点关注产品在不同用户情绪下的表现，以及它的行为对用户情绪的影响。为此，他可以设计几个快乐、焦急、紧张、恼怒的情景，然后在这些情绪氛围中扮演用户，尽力模仿用户的行为模式。一个生动的故事能够帮助测试人员找到用户的“感觉”，很自然地模拟出用户的操作，并切身感受到软件的行为对使用者的情绪有何影响。

在实施情景测试的过程中，测试人员也可以利用一些漫游测试方法，让它们来指引故事的发展，以拓宽测试的视野。以下是一些典型的例子。

- **测试人员利用竞争者漫游来设计故事**。他漫游竞争产品，并收集大众对它的评价，接着设计一组故事来展示了它的优点和魅力。然后，测试人员尝试用被测产品去再现这些故事。也许被测产品提供了不同的功能集合，但是它应该有能力用自己的方式去完成故事，并获得令人满意的结果。该测试比较了两个竞争性的产品，检查了被测产品的实际竞争力。此外，测试人员还可以编写一些故事来展示竞争对手的不足，然后用被测产品来尝试这些故事，以检查被测产品能否克服这些困难。
- **情景测试可以利用快递漫游来追踪一个重要业务对象的生命周期**。例如，订单是在线交

易的核心对象，从新建到关闭会历经多个状态、涉及多个业务模块。测试人员可以设计一个故事以完整地覆盖订单的状态变迁，然后采用快递漫游来跟随订单走遍交易系统的各个模块。在测试过程中，故事提供了业务背景和用户目标，相比单纯的遍历模块，更切合产品的实际使用情景，更有可信度和说服力。

- **情景测试可以使用垃圾车漫游来覆盖一组对象**。垃圾车漫游建议测试人员选择一组对象，如所有与文件输出相关的界面元素、所有支持的文件输出格式等，然后用尽可能短的路径遍历这批对象。测试人员可以设计几个故事来提供遍历路径，这不但覆盖了对象，还通过故事所表达的业务场景测试了它们之间的交互。构思一个有意义的故事会推动测试人员理解对象之间的联系，从业务和技术等角度考虑它们的协作，有助于更好的测试设计。

除了基于覆盖的漫游方法，测试人员在执行情景测试的过程中，还可以使用一些基于风险的漫游方法和快速测试方法，对软件功能进行短暂但猛烈的攻击。这有助于发现一些在压力下才会暴露的问题。有时，此类测试会开辟出新的情节，让测试人员设计出更多的故事。

5.6.4　肥皂剧测试

肥皂剧测试是情景测试的特例。其发明者Hans Buwalda建议测试故事应该像电视肥皂剧一样精彩动人[Buwalda04]。具体而言，一个测试故事应该具备以下特征。

- **源于真实生活**。好的肥皂剧来源于生活，即便剧情难免夸张，但是其场景和细节充满真实感，让观众感到亲切可信。测试故事也要基于用户的实际场景，聚焦可能真实发生的故事。这并不是束缚测试人员的手脚，而是建议他多了解用户和业务。深入地研究用户领域、挖掘业务细节，可以获得深刻的情景，揭示出产品的深层次错误。
- **夸张**。肥皂剧之所以好看，是因为它夸大了生活中的喜怒哀乐，用戏剧性的手法让故事曲折动人。测试故事也需要用戏剧化的情节去考验软件。测试人员用复杂的数据、困难的问题、沉重的负载来挑战软件，因为现实世界是如此复杂，那些看似夸张的情况本来就是生活的一部分。
- **浓缩**。肥皂剧将大量的人物和场景压缩在45分钟的时间内，并展开多个支线情节，让它们相互交织并彼此推动。肥皂剧测试也需要在一个故事中浓缩多项业务，通过同时展开多个复杂的情况以检查软件的设计。软件也许可以分别处理每一项业务，但是当多个业务同时提交且相互牵扯时，软件的设计缺陷可能会让用户一筹莫展。
- **乐趣**。肥皂剧的最大吸引力是带给观众乐趣，而充满乐趣的测试故事能让测试人员热情高涨，思维活跃。软件测试是高水平的智力活动，需要测试者全力投入。有趣的测试情景不但可以帮助测试人员全身心投入，而且能够激发测试人员的灵感，让他们发展出更好的支线和细节。

根据肥皂剧测试的特征，可以用更夸张的场景将5.6.1节的故事改写为下面的样子。



Brian Crum是Contoso公司的总裁。他从欧洲的总部飞到纽约主持新产品的发布会。他原打算周一飞回欧洲，在周三主持全体员工大会，谁知周一当天暴风雨突袭纽约，导致所有航班被取消。无奈之下，他突然想到可以用Lync进行视频会议，便吩咐秘书Bonnie安排一个全体欧洲员工参加的Lync会议，顺便也邀请亚洲的员工一起参加。于是，Bonnie安排了一个跨3个时区、包含1000多个与会者的视频会议。会议当天，Brain和1000多个员工陆陆续续登录到Lync会议中。Brain看着Lync中"一望无际"的与会者列表，感到很满意，于是在即时消息的窗口键入"hi"并发送。在几秒内，有近百名员工回复问候，即时消息的窗口飞快地滚过他们的消息。Brian转头对在另一台电脑上参与会议的Bonnie说："我要开始会议了，请帮我做记录。"然后，他取消麦克风的静音，打开摄像头，启动PowerPoint分享，几秒种后他的视频和幻灯片图像传递到欧洲和亚洲。在Brian演讲的过程中，几千名员工都可以看到他的视频和分享的幻灯片。因为通信距离跨越大洋，视频和幻灯片动画有一些延迟，但是总体还算流畅。演讲结束后，Brian将演示权赋予在欧洲的副总裁Alex，让他报告公司在欧洲运营的情况。很快，Brian和其他与会者看到Alex的视频信号，Alex分享了他的第二个屏幕，用Excel的Power View展示了多幅动态图表，介绍了欧洲业务在过去一个季度的变化。流畅的视频很好地展示了动态图表的动画效果，Alex的报告很成功。然后，Alex将演示权转移给亚洲的副总裁Ken，请他介绍亚洲业务的情况。当Ken结束报告，Brian重新主持会议，他宣布进入Q&A环节，请与会者提问。于是，有几十个人开始键入问题，很快即时通讯窗口就翻过去好几屏。Brian只好随机挑选几个回答。在回答过程中，Brian、Alex、Ken轮流发言，并不时地相互补充。与会者可以同时看到他们的视频，听到他们的声音。终于，大会临近尾声，Brian提示大家此会议有共享的OneNote会议纪要（由Bonnie记录），与此同时他、Alex和Ken分别上传了会议所使用的幻灯片和数据文件，供大家下载。在感谢大家参与后，Brian结束了Lync 会议。

除了涵盖原故事所有的测试点，以上故事还测试了更复杂的情况。

- 在Outlook中创建跨时区的Lync会议。
- 在上千人的会议中，通过视频、语音、文字等进行交流。这给Lync服务器集群带来了巨大的工作负载，根本性地改变了测试情景。在低负载情况下可以通过的测试有可能在这样的负载下失败。
- 测试了近百人同时输入文字的情况。
- 测试了多人同时说话的情况。
- 测试了多人同时上传数据的情况。

在实际测试过程中，测试人员还可以引入更夸张的测试想法。例如，他可以模拟几十个人同时说话或十几个人同时分享视频的情况，以测试这些情景对Lync服务器和客户端的影响。这些高负载的场景将暴露出软件架构的局限，从而有助于研发团队更好地评估和优化Lync在大型会议中的表现。

5.6.5　虚拟业务

在情景测试中，创建并运维虚拟业务是一项常见的测试方法。例如，测试数据库产品时，测试人员可以基于被测数据库开发一个库存管理系统；测试B2C在线商城时，测试人员以电商（卖家）的身份搭建一个在线专卖店，并使用虚拟买家模拟出上架、下单、出货、退货、补货等业务行为；测试报表软件时，测试人员用被测产品制作本部门的季度营收报表，并发布给测试小组成员审阅。运营虚拟业务，可以帮助测试人员考察产品在实际业务环境中的表现，不但可以测试到重点的业务和情景，还可以覆盖一些不经常执行但是很重要的操作，例如数据库的备份与恢复、在线数据的导出与导入、报表的共享与反馈等。

除了情景测试的一般特性外，测试人员还可以参考如下建议，使基于虚拟业务的测试过程更加有力。

- **业务主题可以戏剧化**。测试人员可以选择一个有趣的主题作为业务的目标。例如，测试PowerPoint时，测试人员以“吸血鬼攻击完全防备攻略”为主题，创建一个幻灯片文档。这个主题看似“无稽”，但有潜在的正面作用。一方面，它暗示测试是“无约束”的，让测试人员用更发散的思路去设计测试，另一方面，互联网上有大量此类主题的动漫、电影和资料，测试人员很容易获得相关素材来创作幻灯片。在根本上，“有趣”是一个重要的测试推动力，它激发了测试人员的兴趣，引发了更多的测试想法。
- **工作成果要追求高品质**。测试人员应该用高质量标准去要求测试过程产生的业务成果。如果测试结果是产生一份PowerPoint幻灯片，那么该幻灯片应该是一份可以被公开发布的、在广大观众中获得好评的优质文档，它不但拥有丰富的内容，还具备精致的排版。高质量标准意味着强力的约束，而真实用户在编辑文档时也面临多种现实约束。好的软件会帮助用户去应对约束，而不是让他们感到挫折，这是情景测试的检查重点。此外，高质量标准要求测试人员对文档的内容和外观进行精确的控制，这会让他访问软件的细部功能，从而促进了测试的“深度”。
- **测试手段要多样化**。一个用户往往以固定的模式去使用软件，所访问的特性集合也是有限的，但是一组用户会覆盖软件的大部分特性。为了提高测试的“广度”，测试人员应该时刻提醒自己去使用尚未尝试的特性。例如，在测试PowerPoint时，他可以考虑在一个页面中使用尽可能多的特性，并确保最终效果是美观一致的。这要求他综合并协调多个特性的效果，通过精细地控制来组织多种页面元素。这些测试会在一个局部“浓缩”多个功能，有助于发现功能交互的细节问题。
- **业务数据要具有真实性**。许多时候，测试人员很难预测用户会使用何种数据，自行设计的测试数据很可能不能覆盖重要的用户情景。为了提高测试数据的质量，测试人员需要尽量收集用户的业务数据，通过技术手段抹去用户的私密信息后，将数据用于测试。最简单的方法是在虚拟业务中直接使用这批数据，让产品提前经受真实数据的考验。在此基础上，测试人员分析用户数据的特征，总结出它的数值分布和变化规律，然后将获得

的知识用于设计新的测试数据。例如，测试人员分析在线系统的运营日志，分析出每月的业务量呈现稳定增长的态势。于是他根据当前趋势，估算出新版本上线时的业务量和可能的峰值，然后构造数据来测试新版本。“真实”的情景总是在变化的，不存在一成不变的测试情景。测试人员需要关注变化的业务领域，并及时调整测试策略。

虚拟业务是相对复杂的测试，往往需要测试小组投入大量的人力和时间。如果仔细设计并认真执行，它能提供其他测试方法难以获得的重要信息。特别是，它可以揭示出产品的问题和局限对用户的业务有何影响，从而更真实地评估出产品的价值。

5.7 多样地选择测试技术

面对多种多样的测试技术，测试人员的任务是选择合适的技术应用于当前的项目。对此，我建议测试人员参考Cem Kaner、James Bach和Bret Pettichord在《软件测试经验与教训》一书中关于测试策略的论述[Kaner01]。本节简介他们的观点，并给予必要的解释。

测试策略是一组指导测试设计和测试技术选择的想法。它通常概述了测试目标和实施方法，并提供了技术选择的依据。例如，开发团队对代码进行了大量重构，他们将新的构建提交给测试小组，希望发现重构引入的缺陷。对此，不同的测试小组会采用不同的测试策略。

测试策略1：测试小组安排3个测试人员进行检查。一个测试人员运行所有自动化测试用例，并调查失败的测试用例。一个测试人员执行功能漫游，检查所有被修改的和受影响的功能，在此过程中执行一些快速测试，以探测典型错误。一个测试人员利用测试小组积累的用户数据做系统测试，以检查产品能否妥善地完成用户任务。

测试策略2：测试人员将产品部署到预发布环境中，让它处理上一个月的真实业务数据，以检查它能否恰当地使用资源、以预期的速度正确地完成业务。在测试过程中，测试人员用测试工具监控产品对各种资源的使用情况和数据处理的速度，并反复运行测试脚本以检查输出结果是否正确。如果发现资源泄漏、程序崩溃、结果错误等严重问题，他会暂停测试，邀请程序员一起来调查。

测试策略3：测试人员阅读代码变更集，理解重构的意图和手法，针对每一处重要修改，设计并执行一组测试用例。部分测试用例被编码实现，成为定期运行的自动化回归测试的一部分。另一些测试用例是半自动执行，为此测试人员需要增强自制测试工具的功能。余下的测试是手工执行，且无需自动化或文档化。

以上测试策略有不同的内容和重点，适用于不同的项目环境。其共同特征是根据当前测试目标，论述即将执行的测试和所需的资源。在测试执行时，测试人员会运用具体的产品知识，设计出更细致的策略。总体而言，优秀的测试策略具有如下4个特征。

- **产品相关**：好的测试策略总是根据产品的特点进行有针对性的测试。
- **聚焦风险**：测试人员需要分析项目环境、产品元素、质量要求等方面的潜在问题，让测试过程可以探测其中的重要风险。
- **多种多样**：软件、业务和用户行为是高度复杂的，任何单一的测试手段都存在盲点。测试人员需要综合多种差异化的测试手段，从各个角度考验软件。
- **讲求实用**：测试策略受到测试使命、被测对象、测试资源等因素的影响和约束。一个过于简单的测试策略不适合复杂的产品，而一个繁重的测试策略也不适合资源紧张的项目。测试人员要平衡各种因素，让测试策略可以充分利用现有资源，对项目作出尽可能大的贡献。

在选择测试技术时，Cem Kaner等推荐测试人员根据多样化折衷原则，综合运用多种测试技术进行测试。**简单地说，在有限的测试资源约束下，将多种测试执行到“足够好”的水平，要优于将一两种测试执行到完美水平**。长期工作于一个项目有可能让测试人员在不知不觉中进入某个思维定势，以致他总是使用一组固定的测试方法来检查软件。然而，现实世界是开放的，蕴含着无限的可能，固定的测试手段将不能应对变化的用户、业务和竞争环境。为此，测试人员需要经常反思当前的测试策略，将更多的测试技术纳入测试过程。他可以参考5.1节介绍的六要素分类系统或敏捷测试四象限，从所有的分支或象限选择合适的测试技术，以增强不同缺陷的检出能力。

此外，测试专家还敏锐地指出项目的第一个测试策略总是错误的。因为制定初始测试策略时，测试人员对产品和项目还缺乏足够的了解，往往会忽视某些重要的风险，未能安排恰当的测试活动。所以，测试人员应该随着项目进展，持续调整测试策略，通过持续反思“现在我可以测试什么？该如何测试？”来识别新的测试机会。此外，测试人员应该积极利用测试迭代产生的信息来改进测试策略。在迭代过程中，测试人员构建测试模型并产生测试想法，然后通过测试执行来发掘新的知识和信息。根据这些新知，他修改测试模型，激发出新的测试想法，再付诸实践。通过这样的螺旋上升，测试质量会得到持续提升。

5.8　小结

本章介绍了测试技术的分类方法，并讨论了基于启发式方法的一组测试技术。

- 所有测试活动都会涉及范围（测试对象）、覆盖（测试程度）、测试者、风险、活动（测试手段）、评估（测试先知）和目标7个方面。
- 一个具体的测试技术通常只讨论其中的1~3个方面。在实际使用时，测试人员还需要思考其他方面。
- 启发式方法针对复杂问题提出了一种简单的、较可能成功的解决思路。因为任何启发式方法都可能失效，测试人员需要综合使用多个启发式方法。

- 测试人员需要用批判性思考研究他所使用的每项技术，知晓它的优点与不足。
- 掌握一项测试技术需要“重新发明”它，即通过实践加入自身经验和项目特征，以获得新的方法。
- 测试人员可以针对重复出现的测试问题，提炼出解决方案的模式，建立自己的启发式方法。
- 测试先知是识别软件失败的启发式方法。它不能断定软件行为一定正确，只能检查软件运行是否发生特定错误。
- 测试人员需要从不同关系人的角度，设计多个测试先知来检查软件的行为。
- 漫游测试是在特定主题指导下对产品进行探索，不但可以实施基于覆盖或风险的测试，还可以了解产品知识，提供更多的测试想法。
- 快速测试是一组针对特定缺陷的攻击方法，能够快速发现典型的问题。
- 情景测试利用一个或一组情景（故事）作为测试方案，对软件进行探索。好的情景测试基于连贯一致的、可信的、有推动力的、复杂的故事，且容易评估软件是否出错。
- 好的测试策略是与产品相关、聚焦风险、多种多样、讲求实用的。
- 测试人员应该参考多样化折衷原则，综合运用多种测试技术进行测试。
- 测试人员需要在测试迭代中调整测试策略和方法，以持续优化测试的价值。

第 6 章

测试开发

测试开发是为了完成测试使命而进行的代码开发与维护活动。它服务于软件项目的总体目标，利用计算机的能力来实现更好的测试。随着计算机性能和软件技术的快速发展，测试开发已经成为最基础、最重要的测试手段之一。为了持续提高测试效率和产品质量，测试人员需要分析当前项目的测试需求，将合适的开发技术以合理的策略应用于测试开发。

测试开发往往要求测试人员在代码编写、调试、维护上投入可观的时间和精力。如果这些投入不能带来相应的回报，测试小组就浪费了原本可用于其他测试活动的资源。为了让测试开发交付应有的价值，测试人员需要建立正确的观念，运用合理的开发方法。为此，本章将讨论测试开发的分类，并介绍一些开发思路和策略，供测试人员在实践中参考。

6.1 测试开发分类

测试开发的对象是测试程序，根据测试程序的用途可以将测试开发大致分为两类。

- **自动化测试开发与维护自动执行的测试用例**。这批测试用例的准备、执行、检验、清理和报告都是自动完成的。自动化测试的典型例子是用xUnit测试框架编写的自动化单元测试[WikipediaXUnit13]。
- **计算机辅助测试开发与维护帮助完成测试任务的工具**。其常见的应用领域包括产品分析、测试环境管理、测试数据生成、软件调试、测试结果分析与管理等。

在本质上，自动化测试是一种计算机辅助测试，即自动执行的测试用例也是帮助完成测试任务的工具。因为测试用例是非常重要的基础工具，是许多团队强烈依赖的质量保障工具，所以它被独立出来做专门的研究。这是合理且现实的划分策略。不过，正因为自动化测试和计算机辅助测试在概念上有重叠，所以一些测试代码兼有测试用例和测试工具的特征。例如，5.3.3节介绍了Harry Robinson测试谷歌地图时所采纳的测试开发策略。Harry的测试程序调用谷歌地图，获得两个地点之间的导航路线，然后用一组规则检查导航路线，以发现违反规则的导航路线。该程序可以看作自动化测试，每一对地点就是一条测试用例，其执行、检查和报告都是自动完成的。测试小组可以将这批测试用例当做回归测试用例集，通过定期运行来发现代码改动引入的错误。不过，

Harry的程序并不断定某条导航路线出错，而是提交“疑似错误案例”给测试人员做进一步调查。从这个角度，它是一个起“过滤器”作用的测试执行与分析工具，通过执行海量的计算，提取出测试人员需要研究的异常情况。

接下来，本章将分别讨论自动化测试和计算机辅助测试的基本实践。虽然它们开发不同任务的测试代码，但是都遵循相同的开发原则，拥有相似的实现策略。

6.2 注重实效的自动化测试

自动化测试，顾名思义，是计算机自动执行的测试。测试人员的开发对象是被计算机自动执行的测试用例。在许多项目中，自动化测试用例组成自动化测试套件，被周期性地执行，以持续提供代码的质量信息。因为自动化测试所带来的影响是如此深远，软件专家Martin Fowler指出对于许多项目，测试已经步入编程的前沿和中心位置[Meszaros07]。

为了实现更有效的自动化测试，本节讨论自动化测试的目标和一批值得参考的实践方法。基本想法是根据项目语境，通过测试迭代开发出注重实效的自动化测试。

6.2.1 自动化测试的基本策略

为了取得自动化测试的成功，测试人员需要明确目标。从根本上说，自动化测试的目标是拓展测试小组的能力，帮助他们更好地服务于软件项目。即测试代码并不是自动化测试的目标，在整体上优化测试效率和提高测试效果才是测试小组追求的对象。为此，测试小组需要基于项目语境制订整体的测试计划（包含测试策略、保障条件、测试产出等），将自动化测试作为实现测试策略的重要工具加以有效利用。

由于自动化测试服务于项目整体的测试策略，所以测试开发策略也需要符合优秀测试策略的4个特征：产品相关、聚焦风险、多种多样和讲求实用[Kaner01]。

第一，自动化测试应该切合当前产品。近年来，软件开发与测试技术快速发展，这让测试人员可以从多个优秀的测试框架中进行选择。此外，测试人员还可以利用最新的开发技术来构建自己的测试框架。在选择和创新的过程中，技术选型的基本出发点是自动化测试应该符合产品和项目的特征。这要求测试人员评估当前项目，谨慎地选择实践方法和开发技术，并创造性地将它们应用于产品测试。

语境驱动测试认为“在特定语境下存在好的实践，但不存在最佳实践”[ContextDrvienTesting12]。该原则并非否定现有的开发原则和实践，而是建议测试人员理解并改造已有的方法，以适应项目和产品的特点。例如，“持续集成”是公认的核心开发实践，但是不同的团队会采用不同的具体做法。对于小型项目，一次代码提交便会触发一次完整的构建，并运行所有的自动化测试。

对于大型项目，项目团队也许会每小时做一次构建并运行BVT，然后在每天晚上运行所有的自动化测试。对于超大型项目，做一次完整的构建（包含BVT）需要几天的时间，所以测试小组只会选择某些构建运行所有的自动化测试，自动化测试的运行周期可能是1~2周。可见，好的思想在不同环境中会发展出不同的流程和方法，“因地制宜”地选择和应用才能真正发挥思想的内在价值。

第二，自动化测试应该聚焦风险，重点解决产品面临的最大风险，而不必强求面面俱到。在战术上，这要求测试人员采取进攻的心态，通过自动化测试去迅速发现软件缺陷，让程序员能够快速地修复缺陷，从而获得平稳且有效率的开发过程。

自动化测试面临的第一个风险来自于它本身。有时，测试人员在编写和维护测试程序上投入了大量的时间和精力，却没有获得相应的测试收益。这浪费了测试资源，减少了其他测试活动可用的时间，给产品质量和项目进度带来了负面影响。此外，没有成效的自动化测试会挫伤测试人员的士气，让他们对自动化测试感到厌倦，以致错过了一些明显的改进测试代码的机会。

为了实施有效的自动化测试，我建议测试小组迭代地实施自动化测试。在每一个迭代中，测试小组分析产品风险和项目风险，列出一组需要解决的问题。然后，选出若干高优先级的、适合自动化测试的问题，作为自动化测试的工作目标。针对这几个目标，测试小组拟定出明确的开发任务，安排具体人员来负责实现。在迭代结束后，测试小组评估目标是否达成，并分析自动化测试是否缓解了风险、解决了问题。评估的目标不是评价测试人员的绩效，而是发现自动化测试的长处与不足，讨论现有自动化策略的可改进之处。评估的结果可以是一页清单，记录了应该坚持的实践、需要避免的陷阱、值得尝试的新方法、亟待解决的问题等。之后，测试小组根据当时的产品风险和上一次迭代的结果，安排新的自动化测试任务。

与基于测程的测试管理（参见3.2.13节和4.2.2节）相似，迭代的自动化测试开发是一种自适应的测试策略，它拥有如下特征。

- **时间盒**。测试人员将一个迭代作为一个时间盒，为目标设定、代码开发和结果评估提供明确的期限和节奏。通过多个时间盒的持续努力，测试人员可以积累多个小成果，以获得强大的自动化测试。时间盒的长短取决于项目的特征，通常2~4周的时间适用于大多数项目。
- **在时间和资源约束下的具体目标**。时间盒的基本元素有两点：明确的时间期限和具体的开发目标。测试人员可以拥有一个长远的开发计划，不过对于当前的时间盒，他需要根据项目需要和可用资源，安排一组切实可行的小目标。小目标让测试人员更加专注于开发，而且完成目标的成就感能够鼓舞他更有信心地工作。
- **聚焦风险**。在设定目标时，测试人员需要着重考虑项目风险。这要求他综合考虑短期内需要解决的问题和长期有价值的任务。在一些项目中，测试小组总是遇到许多“突发事

件”，如产品不能安装、自动化测试遭遇大量失败、严重的缺陷使产品工作流中断等。应对这些问题耗费了测试小组大量的精力，使他们无力做更有长远价值的任务。对此，一个可行的策略是优先解决紧急的问题，并通过高质量的自动化测试、合理的工作流程等，降低此类问题发生的可能和修复代价。即通过一套解决方案稳定开发流程、降低产品风险，以便测试小组有资源做更长远的工作。

- **持续评估**。随着项目进展，产品具备更丰富的能力，也面临不同的风险。所以，测试人员需要在迭代间隙持续评估项目情况，以便让新的自动化测试能够应对以往没有考虑到的问题，将精力投放在对产品开发最有价值的领域。
- **动态计划**。迭代开发的本质是根据变化的情况实施动态计划，并根据计划执行结果做出必要的调整。这对于测试开发尤其重要，因为测试代码强烈依赖于产品代码，测试计划需要符合产品开发的实际进度。此外，测试人员通常会同时执行多项测试任务，无法专注于测试开发，其工作计划也时常被突发性事件打断。迭代开发帮助测试人员处理变化的情况，使得投入的资源可以获得相应的回报。

第三，自动化测试应该在资源允许的范围内尽力拓展测试领域，以提供给多样化的测试。功能测试是常见的自动化测试领域，测试小组会构建自动执行的功能测试用例集，以发现代码变动产生的缺陷。除此之外，自动化测试还可以在许多方面大显身手，获得手工测试难以实现的效果。以下是几个例子。

- **性能测试**。随着项目进展，测试人员需要评估产品的性能是在稳步提高，还是在逐渐下降。对于网络应用，随着用户数和业务量的提高，测试人员需要检查即将发布的版本能否处理正常流量和峰值流量。为此，测试人员会编写代码，模拟出各种工作负载，从而在项目过程中持续检查软件的性能。有些测试小组将多次测试获得的性能数据绘制成趋势图，可以轻易地看出产品的性能变化。
- **压力测试**。测试程序驱使产品长时间地处理大批数据，让它超负荷运转，并监控它所暴露出的问题。这样的测试有时会持续几天，记录大量的产品日志和性能数据。测试工具会分析这些日志和数据，以报告其中值得注意的问题。
- **配置测试**。一些软件需要在多个平台上运行，例如一款流行的智能手机应用往往需要支持十多款手机。手工在各个平台上执行测试费时费力，且容易因为测试疲劳造成缺陷遗漏。测试小组开发出跨平台的测试框架，让同一个测试用例集可以运行在多个目标平台上。这不但节省了人力，还加速了测试过程，有助于快速发现在特定平台上才存在的缺陷。
- **安全性测试**。随着软件技术的发展，寻找（并利用）安全性缺陷的技术挑战快速提高。一些安全性问题的技术难度已经超过普通测试人员的能力范围。为了应对日趋棘手的安全性问题，测试小组会利用已有的测试工具，或让安全性专家主持研发测试工具，通过源代码扫描[Chess07]、模糊测试[Sutton07]等手段，来强力挖掘安全性缺陷。

第四，自动化测试应该讲求实用，测试人员需要根据项目语境选择合适的开发策略。软件产品的差异性很大，不存在普遍适用的测试开发策略。在规划和实现自动化测试时，测试小组要思考各种项目因素来做出合理的设计。常见的考虑因素如下。

- **测试使命**。测试小组通常对测试资源（时间、机器、工具等）和测试人手，有一定发言权，但不能作出最终决策（这是项目管理者的职权范围）。在资源相对固定的情况下，将多少资源和人手投入自动化测试，需要测试领导与项目关系人共同决定测试小组承担的测试使命。如果测试小组需要承担大量的手工测试任务，那么他们就很难拓展自动化测试的范围，只能专注于优先级最高的自动化测试领域。
- **测试人员**。有些自动化测试技术适用于编程经验不多的测试人员，有些则要求测试人员精通某类开发技术。为初级测试人员选择高难度的方法，会让他们难以上手，影响项目进度；为高级测试人员安排过于简单的工具，会限制他们的能力，挫伤其积极性。通常，测试领导应该与资深测试人员协商，共同决定测试小组的测试开发策略，并对测试人员的技能发展做出安排。
- **项目进度**。许多项目的开发具有明显的节奏。在项目初期，产品代码尚未完全就位，测试人员有时间开发一些基础性的测试框架；在项目中期，产品代码逐步交付测试，测试人员需要编写组件级别和系统级别的自动化测试；在项目后期，产品临近交付，测试人员专注于运行测试、报告缺陷、验证修复，只能将少量时间用于开发新的自动化测试。这表明测试小组应该根据项目进度逐步演变测试开发的重点。
- **产品特征**。产品的开发技术和业务领域会影响测试技术的选择。如果产品代码用Java编写，那么测试代码就倾向于能够运行在Java虚拟机上的语言。这简化了测试代码和产品代码的相互调用，并方便测试人员了解被测产品的平台特征和技术特点。如果产品需要处理大量的业务数据，那么测试设计就倾向于用数据库保存产品的输入和输出数据，因为数据库提供了便利的方法来存储、查询、分析和计算大规模数据，令测试实现简洁高效。

总而言之，自动化测试是实现测试策略的工具，测试开发需要服务于整体测试计划。测试小组应该注重实效，迭代地开发有效降低项目风险的自动化测试。

6.2.2　将测试开发视作软件开发

自动化测试是基于产品代码的软件应用，测试开发也是一种软件开发，需要遵循恰当的软件开发过程。测试小组需要分析软件开发的基本元素，以选择合适的测试开发过程。图6-1展示了一种可行的测试开发策略，它包含设定目标和测试开发迭代两个阶段。

首先，软件项目要持续地交付价值才能长远地发展，测试开发项目也是如此。测试小组需要分析项目测试面临的挑战，设定测试自动化的使命和目标，并提出候选的实现策略。以下是一些需要思考的问题。



图6-1　迭代地展开测试开发

- 项目和产品面临什么风险？有哪些重要的缺陷要去发现？
- 当前的测试面临哪些挑战？有哪些重要的测试任务？
- 测试小组用什么测试策略去缓解风险？如何用受限的资源去发现尽可能多的重要缺陷？
- 测试自动化在整体测试策略中起什么作用？它如何与其他测试活动配合？
- 利用测试自动化能否发展出更强大、更高效的测试策略？
- 测试自动化的短期目标是什么？长期目标是什么？
- 选择哪些自动化技术来实现目标？如何评估候选技术？
- 执行哪些任务来达成目标？如何安排任务的优先级？
- 如何评估自动化测试达成了预定目标？
- 引入（新的）自动化测试有哪些风险？如何缓解这些风险？
- 为了取得自动化测试的成果，需要对测试流程和人员角色做哪些安排？
- 测试人员需要做何种思想和技术准备？

获得这些问题的答案需要统筹考虑项目全局，也需要深入研究具体问题。为了获得更全面的理解，测试领导可以和几个资深测试人员一起讨论，共同拟定测试开发的计划文档，以说明自动化测试的目标、任务、策略和资源。然后，测试领导邀请测试小组成员审阅计划，通过收集更多的反馈来改进计划，并使之成为大家的共识。经过集体讨论，测试小组获得了测试开发计划的初稿，用它来作为自动化测试的行动指南。

随着项目发展，测试小组面临的挑战也在不停地变化，原先的测试策略可能不适应新的项目情况。为了让自动化测试持续提供价值，测试小组需要积极主动地调整测试开发计划。一个可行

的方法是采用迭代开发过程，用短的开发周期来逐步构建自动化测试，并动态调整方向。通常，测试开发周期为2~4周，在此期间测试小组执行如下任务。

(1) 测试小组评估上一个周期的测试开发，列出没有完成的目标和需要解决的问题。

(2) 测试小组分析当前的项目情况，列出需要实现的自动化测试和值得研究的开发技术。

(3) 测试小组根据可以用的资源，拟定本次迭代的开发目标，并安排实现目标的一组任务。每一个任务都有明确的负责人、完成时间和交付结果。

(4) 测试小组的成员进行测试编码，并利用沙箱测试、同行评审、持续集成、技术原型等开发实践来保证测试代码的质量。

- ❑ **沙箱测试**：测试人员在一个独立的环境中部署产品、开发并运行自动化测试。独立的环境使测试人员之间的开发与测试不会相互干扰。当测试人员完全控制产品、自动化测试和测试环境时，他可以方便地控制测试运行，以快速地获得测试结果。这有助于迅速发现产品缺陷和编写测试代码。
- ❑ **同行评审**：测试小组需要建立测试代码的评审过程，让一组相对资深的员工评审所有的测试代码。通过评审，资深员工可以将一些代码知识、实现方法和设计模式传授给其他测试人员，并保证新编写的代码符合团队制定的编码标准。代码评审是一个传播知识、建立规范的过程，好的评审过程能够提高整个测试小组的质量意识和编程技能。
- ❑ **持续集成**：项目团队建立持续集成流程，频繁地编译产品和测试代码，并运行自动化测试集，以连续地提供产品代码的质量信息。测试小组应该建立流程，让新编写的自动化测试可以加入持续集成所运行的自动化测试集。通过高频率地运行测试，测试小组不但可以及时发现产品的问题，还可以发现测试代码的不足，从而快速地修复问题，使项目过程更加平稳。
- ❑ **技术原型**：在大规模采纳某种新技术、新框架或新工具前，测试小组用技术原型来评估其可行性。测试人员可以用一个迭代周期来构建技术原型，用它完成真实的测试任务，以便在项目语境中分析新方法的长处与不足，从而确定它能否带来预期的回报。如果有多个相互竞争的技术，测试人员可以用它们构建几个技术原型，来完成同一个任务。通过比较技术原型的表现，测试人员能够作出合理的技术选择。

(5) 测试开发的特殊性在于测试小组不但要开发代码，还要运行代码，提交缺陷报告，并修复测试代码的缺陷。即自动化测试项目包含持续的编码、运行、分析、调试和维护。为此，测试小组应该建立明确的流程和责任，让自动化测试可以持续地运行，让测试结果得到及时地调查，让测试缺陷得到快速地修复，让过时的测试被删除或改写。好的流程让新的自动化测试立即发挥作用，并通过测试迭代逐步提高自动化测试集的质量。

总之，自动化测试是一种重要的投资，测试人员会投入可观的时间和精力。为了获得良好的回报，测试小组需要选用恰当的测试设计、开发技术和管理方法，让它们相互支持、彼此配合。

6.2.3　利用自动化测试金字塔来指导测试开发

为了优化测试资源分配，软件专家Mike Cohn提出了测试自动化金字塔[Cohn09]，测试专家Lisa Crispin将其细化为图6-2的形式 [Crispin09]。金字塔包含3个自动化测试层次，在金字塔之上是手工测试。项目团队应该将更多的精力放在金字塔的底层，编写数量庞大的单元测试和组件测试，为高层的自动化测试奠定坚实的基础。

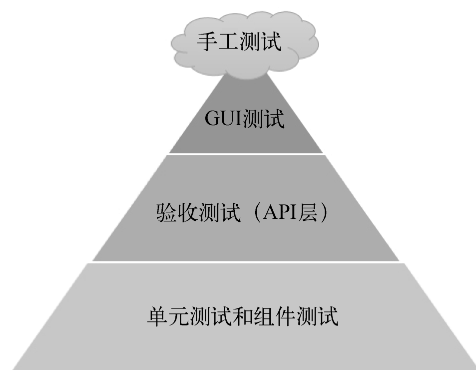


图6-2　测试自动化金字塔

- **金字塔底层是自动化测试的基础，主要包含单元测试和组件测试**。该层次拥有的测试代码最多，是自动化测试投资回报率最高的部分。一般情况下，这部分测试代码由编程小组编写和维护。
- **金字塔中层是面向业务的自动化测试**。它们调用产品提供的编程接口，不依赖于图形界面，所以比较容易开发和维护，拥有较高的投资回报率。通常，测试小组会编写该层的测试代码，开发小组会提供必要的技术支持，例如提供面向业务领域的API。
- **金字塔顶层是少量的基于图形界面的自动化测试**。由于图形界面改动频繁，所以测试代码需要经常修改。另外，基于图形界面的测试代码难以编写、调试和修正，操作系统的偶然事件、用户界面的轻微延迟、测试代码的考虑不周都会导致测试失败，所以拥有高昂的测试维护代价。因此，该层次的自动化投资回报率最低，应该只包含必须使用图形界面才能完成的测试。此类代码通常由测试小组开发和维护。
- **金字塔上方是手工测试**。测试人员在自动化测试的坚实基础上，对尚未覆盖的领域实施手工测试。手工测试并非不使用任何软件工具。测试人员可以选择和开发合适的工具，来辅助测试并提高测试效率。

自动化金字塔是一个概念模型，描述了一种高效的自动化测试策略，但是许多项目的实际情况与之不同。在一些项目中，金字塔处于倒置的状态：程序员只编写很少的单元测试和组件测试，测试人员基于编程接口和图形界面编写测试代码，而且许多代码强烈依赖于图形界面。

这导致了高昂的测试代码开发和维护代价。在另一些项目中，程序员不编写单元测试，测试小组开发的自动化测试也很少，大多数测试需要手工完成。这导致测试小组在每个项目周期都花费大量的时间去手工运行回归测试，压缩了测试新功能的时间，并难以及时发现代码变更引入的回归错误。

面对以上情况，测试小组需要通过持续努力来逐步扭转不利的情况。以下是一些常见的方法。

- **测试小组需要与编程小组商谈，共同制定自动化测试的策略**。最理想的情况是编程小组承诺为新代码编写单元测试，并为测试小组提供足够的编程接口，以简化中层测试用例的开发。其中，单元测试需要达到的质量标准和编程接口的具体内容由编程小组和测试小组协商决定。无论如何，项目团队应该就自动化测试的覆盖范围、不同类型的自动化测试由谁编写和维护、编程小组如何支持测试小组等基本问题达成一致意见。
- **测试自动化要立即见效[Kaner01]**。测试小组选择投资回报率高的区域实施自动化测试。这通常是产品的不稳定区域，拥有频繁的代码变更或大量的软件缺陷。快速的代码变更暗示这部分代码的业务需求比较活跃，是产品的重要价值所在；活跃的缺陷说明这部分代码比较复杂，可能存在业务或技术上的难题。对这些区域建立自动化测试，能够快速地发现新引入的缺陷，让整个项目团队看到自动化测试的价值。
- **测试小组与项目关系人讨论，了解他们最担心的风险和最想知道的信息，然后思考自动化测试能否缓解这些风险或提供相关信息**。与上一点相似，测试小组不是立即实施全面的自动化测试，而是针对高优先级的目标实现自动化测试。通过自动化测试更好地为项目关系人服务，不但完成了测试使命，还让关系人理解并赞赏测试小组在自动化测试上的投入，从而为更广泛的自动化争取更多的资源。
- **在合适的层次和粒度实施自动化测试**。通常，测试小组需要检查软件对最终用户的价值，其拥有的测试资源不允许他们开发所有层次的测试。所以，常见的测试策略是开发面向业务的自动化测试（金字塔的中层），并辅以必要的基于用户界面的测试（金字塔的顶层）。这些自动化测试能够覆盖基本用户情景，为更细致的手工测试提供质量基础。测试人员不必强求所有测试用例都100%自动化[Kaner01]，这不但会带来高昂的测试维护代价，还会压缩实际的测试时间。合理的策略是针对不同的测试任务采取合适的测试方法。在大多数情况下，综合运用手工测试、代码审查、开发者测试、自动化测试等方法会获得更好的测试效果。
- **学习并实践已有的测试开发方法和模式，以快速构建稳定的自动化测试**。*XUnit Test Patterns: Refactoring Test Code*[①]和*Working Effectively with Legacy Code*[②]是该领域的经典著作，值得测试人员参考和借鉴。

金字塔不是一天建成的。测试小组可以从影响大、成本小的自动化测试开始，逐步构建相对

① http://xunitpatterns.com/。
② 中译本为《修改代码的艺术》（http://book.douban.com/subject/2248759/）。

全面的自动化测试用例集合。积小胜为大胜，以稳健的策略去推动测试开发，能够获得短期的成果和长远的发展。

6.2.4 面向调试的测试代码

自动化测试可以视为基于产品代码的二次开发。一般情况下，开发人员会假设所调用的基础性代码是经过充分测试、比较稳定、比较可靠的，但是测试开发却面临截然相反的情况，其调用的产品代码未经充分测试，变更频繁，很可能包含缺陷。因此，自动化测试遭遇失败是“常态”，调查失败原因是测试人员的基本工作。当自动化测试包含大量的测试用例，高效的失败调查能够及时发现产品缺陷，反之低效的失败调查会延迟缺陷发现时间，并占用许多测试时间。

有经验的程序员知道故障调查是一件困难的任务，可能会消耗大量的开发时间，所以他们的设计会考虑软件的可调试性，通过一些特定的功能来降低调试的难度和开销。因为自动化测试的根本任务就是发现并定位产品错误，所以测试人员更要将可调试性作为测试开发的基本因素，时刻加以关注。以下是一些有助于提高可调试性的开发方法。

- **测试代码应该尽可能地简单**。程序员有时为了提高产品的性能，会使用一些复杂的算法和高级的程序库。不过，大多数测试代码并不需要极致的性能，测试人员应该用平实的算法和基础的程序库来构建测试程序。这有助于构建正确的代码，避免复杂逻辑带来的隐藏缺陷。通常，一个测试函数只测试一个具体的功能，一个复杂的测试用例由多个简单的测试函数组成。
- **测试代码应该容易阅读**。测试代码是“可执行的规格说明”，定义了特定操作下被测软件的行为[Features07]。该规格说明应该清晰易懂，让读者很容易理解测试代码做了哪些测试准备、进行了什么操作、检查了什么内容、预期获得怎样的结果等。在理想情况下，读者只需要阅读测试函数就可以理解测试步骤和检查逻辑。这要求测试代码拥有一致的命名、清晰的结构和必要的注释。这些良好的编程实践不但有助于构建清晰的程序，还会节省大量的调试时间，因为理解测试逻辑是发掘失败原因的基本前提。
- **测试代码需要合理的抽象和封装**。许多测试用例会共享一组功能，如启动软件、登录测试用户、检查业务对象属性、关闭软件等。测试人员可以将这批反复使用的代码抽取出来，封装在命名良好的函数和类中。这有助于测试函数的代码聚焦于当前的测试逻辑，而不是纠缠于一些大多数测试都会涉及的操作，从而提高了测试代码的可读性。但是要避免过犹不及，过度的抽象会降低“规格说明”的清晰性，隐蔽重要的操作会让读者错过理解测试逻辑的关键细节。测试人员需要在实践中推敲代码的结构，以获得恰到好处的设计。
- **测试代码应该检查每一步操作的结果，发现问题立即报告**。测试代码应该假定对软件的任何操作都可能失败，所以它会检查所有操作的结果，并通过断言、异常、日志等方式将错误报告出来。第一时间报告错误有助于测试人员理解错误的根源，降低了追踪错误源头的开销。

- **测试代码应该拥抱日志**。测试人员需要用日志记录产品的输入数据、测试的执行步骤、产品的输出结果等信息，以全面描绘测试执行的情况。在理想情况下，测试人员只要对照阅读测试日志和测试代码就可以理解失败原因，而不需要启动调试器。常见的测试框架都提供了功能丰富的日志程序库，并提供了扩展接口来帮助测试人员定制出满足特定需求的测试日志。
- **除了文本日志，测试代码还可以记录其他形式的日志**。例如，有些针对数据库的测试会产生大量的输出结果，将这些结果都写入文本日志并不利于阅读和分析。一个可行的测试策略是将产品输出写入一个独立的测试数据库，以便进一步检查。又例如，在测试执行失败时，测试程序可以屏幕截图，让测试人员了解失败时软件的外观。对于软件崩溃的情况，测试程序可以生成被测进程在崩溃时的内存转储文件，以便测试人员用调试器调查崩溃原因。

代码清单6-1是一个测试Word读取和另存文件的函数。虽然真实的测试代码会更加复杂，但是这个简化的例子很好地说明了如何在测试开发中应用以上实践方法。

- 测试函数TestOpenSave用C#编写，通过调用Word的COM互操作API（Word.Application）来打开指定的文档，然后另存为新的文档，以测试文档读写功能。代码清晰直观，容易理解。
- TestOpenSave提供了两个参数，分别指定需要读取的文件的路径（sourceFilePath）和另存时使用的文件格式（saveFormat），为数据驱动的测试提供了支持。上层的驱动函数可以从外部数据源加载一批文件路径和文件格式，通过反复调用TestOpenSave来读写不同文件和格式。该设计用相对简单的代码支持了多样化的测试。
- 程序库TestLog提供了日志功能，其实现请参考代码清单6-2。TestOpenSave在每一步操作之前都通过TestLog.Comment()来明确记录测试执行到哪里。一旦测试失败，测试人员通过阅读测试日志，就可以知晓哪些步骤已经通过、测试失败具体发生在何处。
- TestOpenSave还用TestLog.Comment记录了所有的输入值（函数参数sourceFilePath和saveFormat的值）。在数据驱动测试中，大量的文件路径和格式被传递给TestOpenSave。记录所有的输入值简化了调试过程，测试人员只要阅读测试日志，就可以了解是哪个文件和格式导致测试失败。
- TestLog还提供了检查功能。TestLog.VerifyEqual比较实际值是否等于预期值，如果不相等，该函数会记录错误消息。TestLog.VerifyTrue判断输入的条件是否为真，如果为假，该函数会记录错误消息。在每一步操作之后，TestOpenSave都会调用这些函数来检查操作是否成功，以第一时间记录问题。这两个检验函数只记录错误消息，不会抛出异常以中断测试执行。在调用结束之后，函数TestOpenSave总能继续执行。
- 程序库TestLib封装了一批辅助函数，提供了可以让多个测试函数复用的功能，其实现请参考代码清单6-3。该程序库和TestOpenSave拥有一致的编码风格，都利用简单明了的指令来操作被测对象，用TestLog来记录操作步骤和检查操作结果。

- TestLib的函数EnsureFileExists和EnsureNoWordProcess以Ensure作为函数名的前缀，以表明它们会检查特定的条件。如果条件不为真，它们会调用TestLog.Fail来记录错误消息并抛出异常。测试代码用这些函数来检查对测试执行非常重要的条件，如果这些条件不成立，测试执行应该中止。

代码清单6-1 测试Word读取和另存文件的函数

```
/// <summary>
/// Test method that opens and saves a document
/// </summary>
/// <param name="sourceFilePath">source document to open</param>
/// <param name="saveFormat">file format to save</param>
public void TestOpenSave(string sourceFilePath, Word.WdSaveFormat saveFormat)
{
    TestLog.Comment("-- Start TestOpenSave");
    TestLog.Comment("sourceFilePath={0}", sourceFilePath);
    TestLog.Comment("saveFormat={0}", saveFormat);

    TestLog.Comment("-- Check test preconditions");
    TestLib.EnsureFileExists(sourceFilePath);

    TestLog.Comment("-- Boot Word");
    Word.ApplicationClass word = TestLib.BootWord();

    TestLog.Comment("-- Open document ({0})", sourceFilePath);
    Word.Document document = word.Documents.Open(sourceFilePath);
    TestLog.VerifyEqual(word.Documents.Count, 1,
        "One document should be opened");

    TestLog.Comment("-- Compute unique target file path");
    string targetFolder = System.IO.Path.GetTempPath();
    TestLog.Comment("targetFolder={0}", targetFolder);
    string targetFile = string.Format("{0}{1}_{2}.{3}",
        targetFolder, System.Guid.NewGuid(),
        saveFormat, this.GetExtension(saveFormat));
    TestLog.Comment("targetFile={0}", targetFile);

    TestLog.Comment("-- Save document");
    object oTargetFile = targetFile;
    object oSaveFormat = saveFormat;
    document.SaveAs2(ref oTargetFile, ref oSaveFormat,
        ref missing, ref missing, ref missing,
        ref missing, ref missing, ref missing,
        ref missing, ref missing, ref missing,
        ref missing, ref missing, ref missing,
        ref missing, ref missing, ref missing);

    TestLog.Comment("-- Check saved document");
    TestLog.VerifyTrue(document.Saved, "Document should be saved");
    TestLog.VerifyEqual(document.FullName.ToLower(), targetFile.ToLower(),
        "File paths should be same");
    TestLib.EnsureFileExists(targetFile);
```

```
        TestLog.Comment("-- Close Word");
        TestLib.CloseAllDocumentAndQuit(word);
    }
```

代码清单6-2　日志程序库TestLog的实现

```
static class TestLog
{
    enum LogLevel
    {
        Comment,
        Warning,
        Error
    }

    public static void Comment(string format, params object[] parameters)
    {
        Log(LogLevel.Comment, format, parameters);
    }

    public static void VerifyTrue(bool condition, string message)
    {
        if (!condition)
        {
            Log(LogLevel.Error, message);
        }
    }

    public static void VerifyEqual(object actual, object expected,
        string message)
    {
        if(!actual.Equals(expected))
        {
            Log(LogLevel.Error,
                "The actual value ({0}) does not equal the expected value ({1}),"
                + "message: {2}",
                actual, expected, message);
        }
    }

    public static void Fail(string format, params object[] parameters)
    {
        Log(LogLevel.Error, format, parameters);
        throw new Exception(string.Format(format, parameters));
    }

    private static void Log(LogLevel level, string format,
        params object[] parameters)
    {
        // 将错误消息广播给日志的监听者
    }

    // ...
}
```

代码清单6-3 程序库TestLib提供了可复用的函数

```
static class TestLib
{
    private static object missing = Type.Missing;

    public static Word.ApplicationClass BootWord()
    {
        TestLog.Comment("-- Boot Word process and show its window");

        var word = new Word.ApplicationClass();
        TestLog.VerifyEqual(Process.GetProcessesByName("winword").Length, 1,
            "There should be one Word process");

        return word;
    }

    public static void CloseAllDocumentAndQuit(Word.ApplicationClass word)
    {
        TestLog.Comment("-- Start CloseAllDocumentAndQuit");
        object doNotSaveChanges = Word.WdSaveOptions.wdDoNotSaveChanges;

        TestLog.Comment("-- Close all documents");
        int beforeCloseCount = word.Documents.Count;
        TestLog.Comment("Before close, Word document count={0}",
            beforeCloseCount);

        while (word.Documents.Count > 0)
        {
            word.ActiveDocument.Close(ref doNotSaveChanges,
                ref missing, ref missing);
            int afterCloseCount = word.Documents.Count;

            TestLog.Comment("After close, Word document count={0}",
                afterCloseCount);
            TestLog.VerifyTrue(beforeCloseCount - afterCloseCount == 1,
                "One document should be closed");
            beforeCloseCount = afterCloseCount;
        }

        TestLog.Comment("-- Quit Word process");
        word.Quit(ref missing, ref missing, ref missing);

        System.Threading.Thread.Sleep(2 * 1000); // 等2秒，以待Word进程退出
        EnsureNoWordProcess();
    }

    public static void EnsureFileExists(string filePath)
    {
        if (!System.IO.File.Exists(filePath))
            TestLog.Fail("file ({0}) does not exist", filePath);
    }

    public static void EnsureNoWordProcess()
```

```
        {
            int wordProcessCount = Process.GetProcessesByName("winword").Length;
            if (wordProcessCount != 0)
                TestLog.Fail("There should not be Word processes");
        }
    }
```

虽然编写面向调试的代码需要测试人员付出额外的努力，但是这能获得长久的收益。大多数自动化测试会运行多次，一些测试用例甚至会运行多年。良好的可调试性能够大幅度降低失败调查的开销，帮助测试人员更快地提交缺陷和修复测试代码的错误。

6.2.5　系统测试的测试开发

随着测试对象的变化，测试开发的需求和方法也会发生相应的变化。例如，单元级别和系统级别的自动化测试就存在显著差异，表6-1概述了它们的不同点。

表6-1　对比单元级别和系统级别的自动化测试

	单元测试	系统测试
对象	• 测试（一个类的）一个函数。 • 测试几个紧密联系的函数（可能属于不同的类）	• 测试系统一个功能。 • 测试几个相互协作的功能。对于大型系统，一条测试用例会覆盖若干子系统
需求	• 测试运行速度快，可以立即提供对代码修改的检查结果。 • 测试用例集能够周密地覆盖被测对象。优秀的测试用例集能提供90%以上的语句覆盖率	• 测试用例的首要任务是检查产品在真实环境中的表现。其测试运行速度通常比单元测试慢几个数量级。 • 测试用例覆盖最重要的业务流程。受到技术和资源的限制，系统测试的语句覆盖率较低
手段	• 根据代码实现设计测试用例，属于白盒测试范畴。 • 为了提高运行速度，好的单元测试不访问文件、网络、数据库、服务等外部资源。 • 通过测试桩、模拟对象等技术来替换被测对象依赖的外部资源。 • 测试框架程序加载测试用例和被测对象，让它们位于同一个进程空间。 • 每一个测试用例会创建自己的测试对象，因此测试用例之间相互独立	• 测试用例对系统进行黑盒测试。有效的测试设计需要了解产品架构、组件功能、输入输出格式等技术细节，因此具有灰盒测试的特点。 • 为了测试产品的真实表现，系统测试会覆盖文件、网络、数据库、服务等外部资源。 • 系统测试通常不使用虚拟对象等测试替换技术。如果使用，被替换对象通常是一个组件或子系统。 • 为了方便测试执行和监控，测试用例和被测产品常常运行在两个进程中。 • 一组测试用例常常共享一个测试对象（通常是被测试进程），之前运行的测试用例可能影响到之后运行的测试用例

一般说来，测试人员工作在组件和和系统级别，其测试会覆盖产品的多个部件和功能，所以他的自动化测试也偏向系统层次，会覆盖产品的多个特性。为了更好地开发和维护自动化的系统测试，测试人员除了参考基本的测试开发方法，还要根据系统测试的特点，采取一些有针对性的实践。对此，本节将讨论几个典型的系统测试的开发实践。

实践1：全自动的产品安装是最基本的可测试性需求

代码清单6-4来自单元测试框架NUnit的教程[NUnitQuickStart12]。测试函数`TransferFunds`测试负责转账功能的函数`Account.TransferFunds`，它获得可测试对象的方法是用new操作符生成转出账户和转入账户。这是典型的单元测试的测试准备方法：测试用例和被测对象运行在同一个进程中，由测试框架加载被测对象所在的程序集，由测试用例安排被测试对象的生命周期。整个过程不需要部署完整的产品。

代码清单6-4　单元测试样例

```
[Test]
public void TransferFunds()
{
    // setup
    Account source = new Account();
    source.Deposit(200m);

    Account destination = new Account();
    destination.Deposit(150m);

    // execute
    source.TransferFunds(destination, 100m);

    // check
    Assert.AreEqual(250m, destination.Balance);
    Assert.AreEqual(100m, source.Balance);
}
```

系统测试的测试对象是完整的软件系统或子系统，因此它会要求在测试执行之前软件被正确地安装到测试环境中。为了达成高效的自动化，整个安装过程需要是全自动且可预先配置的。

- **所谓“全自动”是指测试人员只要执行一个命令，安装程序就可以将产品部署到指定的测试环境中，即“一键部署”**。测试人员需要在项目过程中多次部署产品，如果安装过程要求复杂的手工操作，势必会占用大量测试时间，浪费测试人员的精力。而且，枯燥的手工配置很可能因为疏忽而引入安装错误，而包含错误配置的产品会导致系统测试失败。当测试人员费尽心力地调查测试失败，却发现这是安装错误引起的，会备感挫折。
- **所谓“可预先配置”是指测试人员可以用配置文件或配置脚本指挥安装程序，让安装后的产品符合预先定义的配置**。因为大多数产品的安装需要用户提供一些参数值，可预先配置实际上是全自动安装的前提条件。对于分布式系统的开发和测试，这尤其重要。例如，被测的网络服务需要访问数据库，因此测试人员需要配置它的数据库连接字符串，以指向恰当的测试用数据库。可预先配置的安装让测试人员在配置文件中指定数据库连接字符串，令安装后的系统使用测试人员指定的数据库。
- **在全自动和可预先配置的基础上，项目团队可以用一组配置文件在所有环境中部署产品**。例如，配置文件A将产品部署到测试开发环境，该环境通常用一台机器运行多个服务和数

据库；配置文件B将产品部署到功能测试环境，该环境通常包含多台计算机以模拟出产品环境的特征；配置文件C将产品部署到预发布环境，该环境与产品环境拥有相同的拓扑结构和相似的数据；配置文件D将产品部署上线。可预先配置的全自动安装简化了测试环境的搭建，提高整个测试开发与执行的效率。此外，用相同的机制在测试环境和产品环境中部署产品，不但能够尽早地发现安装程序的错误，还可以模拟出产品升级、补丁上线等情景，对于提高安装程序的质量很有帮助。

在大多数团队，安装程序由编程小组负责开发和维护。在项目之初，测试小组、产品经理和编程小组需要共同协商，让安装程序可以支持全自动安装。最佳结果是全自动且可预先配置的安装成为第一等级的功能需求，被写入规格说明和设计文档。可接受的结果是编程小组同意向测试小组提供以下测试支持。

- 安装程序可以部署完整的系统。
- 因为安装失败会阻碍后续测试，安装程序的缺陷会获得最高优先级。
- 当一项功能被提交给测试小组时，该功能可以被安装程序自动部署。
- 如果测试人员认为安装程序需要接受额外的配置参数，产品经理、程序员、测试人员协商决定是否或如何支持该配置参数。

产品安装是一项容易被忽视的工作项。当产品经理思考产品特性、程序员构思代码实现时，他们很可能会忘记正确的安装是交付任何产品价值的前提。作为最早使用产品的人，测试人员应该明确提出全自动安装是一项重要的可测试性需求，需要在整个项目周期获得第一流的支持。

实践2：使用单独的进程监控测试运行

在运行代码清单6-4所示的单元测试时，测试程序（nunit.exe）加载测试函数（`TransferFunds`）和被测类（`Account`）的程序集，在同一个进程中运行测试程序和被测代码。与单元测试不同，系统测试通常会启动多个进程：运行测试用例的进程和被测产品的进程。例如，代码清单6-1所示的系统测试运行在测试框架进程中，其测试对象Word运行在另一个进程中。

在多进程的测试中，被测进程可能发生死循环、死锁、忙等外部资源等情况而失去响应。这时，等待被测进程输出的测试函数也会陷入“假死”状态。对此，测试程序需要有办法终止当前测试用例，记录错误并重启被测进程。一个常见的做法是测试程序启动一个线程来计时，一旦该线程发现测试函数的执行时间超过预定值，就会向执行测试函数的线程发出终止消息。在收到终止消息后，测试执行线程会退出当前测试，并让测试清理代码重置测试环境。

对于复杂产品的系统测试，一个独立的监控进程（甚至一组监控进程）可能是更健壮的测试设计。当测试代码进行复杂操作时，它也可能陷于死循环、死锁等错误中，使得整个测试进程失去响应。此时，运行在测试进程内部的计时线程和测试清理代码可能没有机会运行，使得整个自动化流程无法发现测试已经停滞并进行自我修复。对于这种情况，测试人员可以考虑图6-3所示

的架构，用一个监控进程去监视并管理测试进程和产品进程。它一旦发现某个进程失去响应或运行超时，就记录错误日志并终止该进程。如果是产品进程被终止，测试进程会发现产品进程退出，它会执行测试清理代码，并启动下一条测试。如果是测试进程发生问题，监控进程通常会终止所有进程，并重启测试进程以执行下一条测试。通常，监控代码相对简单，其发生错误的概率低于其他程序，有助于实现稳定的故障恢复和测试重置。

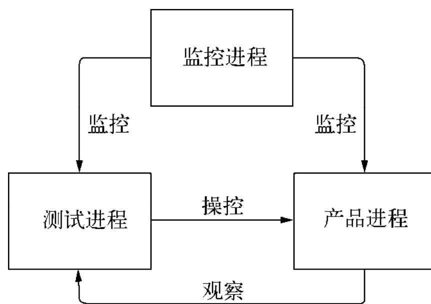


图6-3　监控进程

实践3：采用多种技术记录测试执行过程和被测产品表现

如代码清单6-4所示的单元测试利用断言（`Assert.AreEqual`）和异常来报告软件错误，并将这些信息写入测试日志。因为单元测试目标明确、代码短小，程序员参照测试代码阅读测试日志就以大致了解测试代码的执行情况。通常，他根据断言或异信息就以定位出错的测试语句，向前浏览一段测试代码就能够了解出错的原因。

系统测试的流程较长，覆盖产品的多个功能和部件，可能暴露出多种不同类型的缺陷。因此，测试人员应该用多种方法来监控产品和记录测试执行信息。一方面是提供更多的信息，以便于错误调查，另一方面是从多个角度检查软件，从而识别软件缺陷。以下是一些常见的实践方法。

- 采用6.2.3节介绍的测试开发方法，通过日志函数来记录测试参数、测试步骤、产品输出等信息，让测试日志可以提供测试执行和产品状态的详细信息。
- 采用6.2.3节介绍的测试开发方法，在每一步操作结束后，检查产品的状态和输出，一旦发现问题，就记录错误或抛出异常。这有助于尽早发现不符合预期的异常情况，使得暴露错误的地点尽可能地接近错误源头。
- 通过屏幕截图来捕获产品的状态。对于拥有图形界面的软件而言，一幅屏幕截图可以提供其状态的快照，能够提供许多细节信息。测试人员可以选择或开发支持屏幕截图的测试框架。该框架为测试开发提供了屏幕截图函数，并且会在测试失败时自动截取当前屏幕，将图片保存在测试日志目录。利用屏幕截图函数，测试人员可以在关键操作之前或之后截取屏幕，以记录一组产品的快照。在调查测试失败时，测试人员打开测试日志目录，浏览这组屏幕截图即可以了解产品对测试操作的反应。在许多情况下，这批图片将提供测试失败的重要线索。

- 当测试程序（或监控程序）发现产品处于异常状态时，它调用操作系统提供的API生成产品进程的内存转储文件。该文件可以帮助测试人员了解产品状态的细节，提高故障诊断的效率。此外，测试程序可以修改操作系统配置，让操作系统自动生成崩溃进程的内存转储文件。在测试结束时，测试程序将新生成的内存转储文件移动到测试日志目录，以供测试人员分析。
- 对于长时间运行且不重启产品的系统测试，测试程序需要持续监控产品的性能指标和资源占用情况。常见的性能指标包括产品对用户操作的响应时间、完成特定任务的时间、在单位时间内完成业务的数量等；常见的资源对象包括CPU、内存、句柄、网络带宽、数据库连接等。如果发现产品出现速度迟缓、资源占用过多、资源泄漏等问题，测试程序需要记录错误信息，并保存内存转储文件和相关性能数据。这有助于发现一些长时间测试才能暴露的问题。

实践4：利用关系型数据库存储测试数据和产品输出

如果系统测试需要庞大的测试数据或产品会产生大量的输出数据，测试人员可以考虑用关系型数据库来管理这些输入或输出数据。以下是两个案例。

案例1：订单处理服务A是整个工作流的一部分，它从订单数据库中读取订单，对各个字段进行计算，然后将计算结果写回订单数据库，供其他服务继续处理。在产品环境中，该服务每天会处理大量的订单，产生许多计算结果。测试人员开发了多种自动化测试，从多个角度来测试它。其中一种方法是从数据库的角度来测试服务A，其架构如图6-4所示，其执行过程如下。

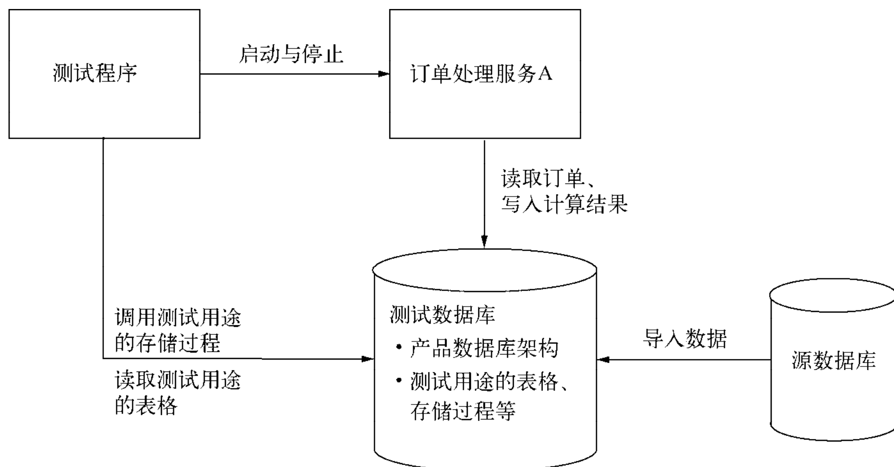


图6-4　用测试数据库来测试订单处理服务

(1) 测试程序调用测试人员编写的数据库创建脚本，生成名字唯一的测试数据库。该数据库的架构包含产品数据库架构以及一组测试用途的表格和存储过程。

(2) 测试程序调用测试数据库的存储过程ImportOrder，将一批订单数据从源数据库导入到测试数据库。源数据库的订单包含产品数据库的真实数据和测试人员制造的测试数据。

(3) 测试程序修改服务A的数据库链接字符串，让它指向测试数据库。

(4) 测试程序启动服务A。

(5) 服务A持续从测试数据库读取订单，并将结果写回测试数据库。

(6) 测试程序调用测试数据库的存储过程CheckProgress，以检查尚未处理的订单数量。如果服务A没有在规定时间内完成所有订单计算，测试程序会记录错误信息，并终止测试。

(7) 在服务A完成了订单计算之后，测试程序调用测试数据库的存储过程CheckResult，以检查数据库中的计算结果。该存储过程调用一组存储过程，从多个角度分析服务A的输出。如果它们发现了错误，会将错误信息写入表格TestErros。

(8) 测试程序读取表格TestErros。如果表格为空，测试程序记录测试通过。如果表格不为空，测试程序记录测试失败，并将测试数据库的连接字符串写入测试日志，供测试人员分析。因为测试程序每次会创建不同名字的测试数据库， 所以之后的测试执行不会删除或修改之前的测试数据库。通常，测试数据库会在几天后被一个定期执行的程序删除。

案例2：测试人员将一个性能测试用例加入每日测试。该测试用例覆盖了一个典型的用户情景。在测试用例的最后一步，它会检查测试执行的用时。如果用时超过规定的时限，它会记录错误信息。无论测试是否失败，它都会将被测产品的版本号、测试用例编号和用时写入一个性能数据库。每两周，一个程序会读取性能数据库，生成报表发送给测试人员。当测试人员读到如图6-5所示的曲线图时，他可以快速了解产品的性能变化趋势。对于编号为201的测试用例，各个版本的用时大致相同，说明产品在这个用户情景中的速度表现比较稳定。对于编号为202的测试用例，版本10.05的用时明显高于之前版本，暗示该版本包含一些代码变更，使得产品速度剧烈下降。即便版本10.05的用时没有超过规定的时限，测试人员也可以花一些时间调查性能下降的原因，以更好地理解产品代码的变化。

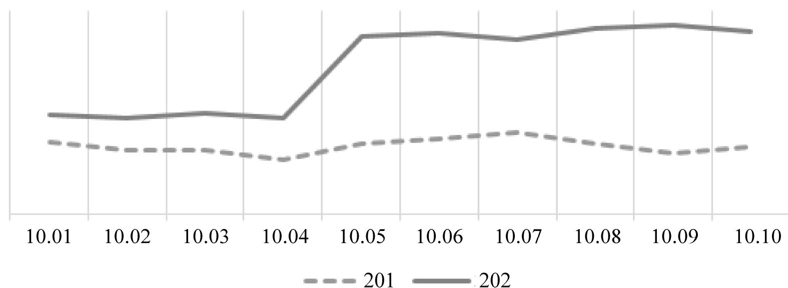


图6-5　性能测试报表

由以上两个案例可以看出数据库技术在系统测试中可以产生一些积极作用。

- **如果产品系统包含数据库，那么可以考虑在测试环境中使用测试数据库来模仿产品数据**

库，以测试访问数据库的软件。测试数据库提供与产品数据库相同的架构（表、视图、函数、存储过程等），从而“无缝”地融入被测系统。它还包含一组测试用途的表、函数、存储过程、触发器等对象，以便在测试过程中检查对数据库的读写。此时，测试数据库是一种模拟对象，它既是被测系统的一部分，承担着特定的业务任务，又是自动化测试的一部分，承担着测试系统的工作。

- **数据库提供了强大的数据存储和处理能力**。利用SQL语言，测试人员能够方便且高效地查询、修改、生成大量的数据。这对自动化测试的数据准备、结果检查、报告生成等任务很有帮助。
- **利用数据库可以方便地分析数据**。在测试设计、故障诊断等任务中，测试人员都需要分析产品所读取和产生的数据。如果这些数据存储在数据库中，测试人员可以用SQL查询去灵活地探索它们，通过反复地查询、分析、再查询来理解数据状态，从而更好地把握整个系统的行为。
- **利用数据库可以方便地生成报表**。目前，有多种成熟的技术可以读取数据库，并生成各种图表。测试人员可以采纳合适的技术，来分析测试数据或测试结果，并获得数据的可视化表示。这有助于测试人员快速理解数据，发掘其中的信息。

实践5：恰当地利用基于图形界面的“捕获回放”技术

在软件测试中，捕获回放是指用工具捕获外界对软件的操作，并记录为测试脚本，然后根据脚本去回放事件来操控软件，并检查软件的反应。基于图形界面的捕获回放技术记录用户对软件的输入事件，包括键盘输入、鼠标点击、触摸手势等，并通过回放这些事件来复现操作流程。基于该技术，测试人员可以将手工的测试过程录制为测试脚本，然后重播该脚本来检查软件的行为与录制时的行为保持一致。

基于图形界面的捕获回放位于自动化测试金字塔的顶层，拥有较高的测试维护代价，并不适合作为回归测试在项目中反复运行。其主要技术局限如下。

- **录制所得的脚本的可读性较差**。它只是忠实地记录了手工测试的动作（常常包含一些冗余的动作），没有体现出测试的意图。因此，理解测试脚本是一项困难的任务。
- **录制所得的脚本依赖于录制时的软件界面**。当软件界面发生变动时，脚本需要被修改或重新录制。在许多项目中，软件界面会快速演化，这显著提高了修正脚本的频率。
- **为了完善测试逻辑，测试人员会修改录制的脚本，以增加一些额外的操作命令或检查指令，而重新录制的脚本不包含这些额外的命令**。测试人员需要在新脚本中再次添加它们，这提高了测试维护的成本。
- **基于图形界面的测试常常受到各种因素的干扰，而不够稳定**。例如，测试用例向软件发送英文字符串，以提供字段值。在测试执行时，中文输入法因为某种原因被启动，于是实际输入的字符串是中文字符串。软件发现输入字符串的格式不正确，于是报告操作失败。如果回归测试产生大量此类虚假失败，测试人员将因为调查和修复它们耗费大量的时间。

虽然基于图形界面的捕获回放不适合反复运行的回归测试，但它适用于一些短期的测试任务。测试人员可以利用捕获技术来快速生成测试脚本，然后用回放技术来重复播放测试脚本，以完成一些操作密集的测试任务。例如，测试人员需要评估被测网站的性能，他采用如下步骤实施测试。

(1) 走访产品经理，了解网站需要支持多少并发会话，并确定一组典型的用户情景。

(2) 走访实际用户，与他们讨论并完善这组使用情景，并确定每个情景的使用频率。

(3) 根据前两步的结果，明确基本测试需求：该网站最多支持500个并发会话，拥有6个典型的用户情景。

(4) 选择一个功能稳定的版本作为测试对象，并搭建测试环境。

(5) 使用捕获回放工具来录制这6个情景。对于每个情景，他先手工操作一遍，然后回放录制所得的脚本。如果脚本不能准确再现手工操作的流程，他会调试并修改脚本，或者重新录制一遍。因为他使用的捕获回放工具比较成熟，获得一个稳定的脚本只需15~20分钟，所以他只用两个小时就生成了所有脚本。如果通过编码来实现这些脚本，他可能需要1~2天的时间。捕获回放工具提高了测试开发的效率。

(6) 用一个虚拟用户来分别运行6个脚本，获得它们在单用户情况下的基线性能数据。

(7) 用测试工具生成50个虚拟用户，并发地执行这6个脚本。他根据使用频率来分配虚拟用户数，即最常见的用户情景获得最多的虚拟用户，相对少见的用户情景获得较少的虚拟用户。该测试持续10分钟，每个虚拟用户都反复执行了它所获得的脚本。测试人员收集到一批性能数据。

(8) 执行测试，以获得100个、200个、300个、400个、500个虚拟用户的性能数据。

(9) 用测试工具生成600个虚拟用户，以检查超过预期峰值的流量会对网站产生何种影响。

(10) 运行1000个虚拟用户的压力测试，并收集性能数据。

(11) 分析所有性能数据，编写性能测试报告以总结测试发现，然后将报告发送给项目团队。

(12) 根据团队成员对性能测试报告所提出的问题，设计并执行新的测试，以获得答案。

分析以上步骤不难看出，该测试策略回避了捕获回放的基本缺点。首先，测试人员只录制少量的脚本来覆盖典型的用户情景，这降低了意外因素对测试运行的影响。其次，所录制的脚本只针对特定版本，从而不必担心用户界面变更导致的测试失败。再次，下次执行系统级的性能测试时，网站可能拥有新的界面和用户情景，测试人员会放弃失效的脚本，录制新的脚本。由于录制新脚本的开销很低，测试人员无需在意这批脚本的可读性和可修改性。

目前，主流的捕获回放工具对脚本录制、脚本播放、虚拟用户生成、性能数据收集、性能数据分析都提供了较好的支持。测试人员可以在一天内完成第(5)步~第(11)步的脚本生成、测试运行、数据分析和报告编写。这加快了性能测试的反馈速度，提高了性能测试的机动性。对于一些项目团队，基于图形界面的捕获回放技术为此类测试提供了足够好的解决方案。

实践6：为测试执行和失败调查制定明确的工作流程

编写自动化系统测试的目的是为了获得产品的质量信息，所以编写代码只完成了工作的一部分。系统测试比单元测试复杂得多，需要大量的时间来运行，会产生大量且复杂的错误信息。如果疏于管理，测试小组可能无法及时、有效地处理这些信息，也不能快速地修复测试代码的错误。久而久之，测试用例集会包含很多有问题的测试用例，产生伴随大量“噪音”的测试结果，并失去测试人员的信任。为了推动系统测试的良性发展，测试小组需要制定工作流程，以明确地回答以下问题。

- 由自动化系统测试组成的回归测试用例集有哪些？
- 这些回归测试用例集以什么频率运行？
- 谁负责启动这些回归测试用例集？
- 谁负责搭建并维护系统测试的测试环境？
- 谁负责发送测试用例集的测试结果报告？
- 谁负责调查测试失败？是测试编写者调查，还是团队共同调查？
- 测试失败调查的完成标准是什么？
- 测试失败调查需要在什么时限内完成？
- 谁负责修复有错误的测试代码？
- 测试代码修复完成的标准什么？
- 测试代码修复需要在什么时限内完成？
- 如果需要临时打破测试运行、调查、修复的例行流程，应该采取什么行动？
- 如果需要修改测试运行、调查、修复的例行流程，应该采取什么行动？

如果将测试代码视为基于产品代码的应用，测试小组就是该应用的运维团队。测试领导和测试人员应该共同制定出运行、调查、修改、增强自动化测试的流程和制度。只有明确人员责任、工作目标、工作期限、变更方法，才有可能让复杂的自动化系统测试持续地提供价值。

6.2.6　让自动化测试服务于项目

自动化测试的目标是帮助整个团队更好地开发产品。在不同的项目，测试小组会采用不同的自动化策略来达成该目标。一般说来，好的自动化测试能够平滑地融入开发流程，快速、稳定、持续地提供产品的质量信息。本节分享我的一次自动化经历，虽然实践方法只适用于特定的项目，但是仍体现了一些好的思路和经验。

我曾经参与过一个分布式系统的测试。当时，测试小组实践了一种与持续集成[Duvall07] 类似的测试活动： 滚动测试。这是一个全自动的编译和测试过程。在每个小时，测试脚本从源代码管理系统获得最新版本的代码，进行完整的编译。然后，在一台干净的机器上部署新构建的系统，并执行端到端的系统测试。最后，用电子邮件将测试结果发送给项目团队。

滚动测试不是严格意义上的持续集成，不过它发挥了相似的作用。我和同事希望通过它来达成以下目标。

- **尽快地发现集成错误**。开发者在自己的开发环境中工作，他所看到的系统是他最后一次检出的结果，他根据这一版本的“知识”来开发新的功能。虽然两个开发者基于相同的“知识”进行开发，但是不能保证他们所构造的新“理论”是相互一致的。往往，两部分代码都通过了单元测试，但是提交之后却引发了集成错误。滚动测试用端到端的系统测试弥补了单元测试的不足，能够快速提供系统级的测试反馈。
- **节省开发时间**。集成错误往往会阻碍全面的系统测试。例如，模块A不能正确解释模块B的输出，它抛出异常并停止工作。这时，系统测试将无法覆盖模块A及其后续模块的代码。这将严重影响测试效率，甚至使得每日运行的全面回归测试变得毫无效果。滚动测试可以在代码检入之后的一个多小时内捕获到此类严重错误，让开发者能够在第一时间修正错误，清除测试障碍。有时开发者的修正会引入的新的集成错误，滚动测试的安全网会再次捕捉到它，于是我们有机会保证：在下班回家的时候，没有明显的集成错误。
- **发现环境错误**。所有的开发者都应该听从John Robbins的忠告：不要在开发机上使用管理员权限[Robbins07]。不过久经世故的Cem Kaner也说过：“本书介绍的测试，假定的前提是你的同伴现在没有、将来也不会并且也没有必要遵循这些规定。”[Kaner99]由于开发者在开发机上拥有管理员权限，他们安装了各式各样的工具和运行库，我们无法保证所开发出的系统能够在其他环境中以合适的权限正常工作。滚动测试在干净的机器上部署被测系统，能够发现许多在开发环境中无法发现的错误。所谓干净的机器，并不要求重新安装操作系统，只是要求完整地卸载了旧版本的系统。
- **保证被测版本的基本质量**。这使测试团队更有信心进行大规模的功能测试和压力测试。

在确定目标后，我们进行了紧张的测试开发，在很短的时间内完成了编码和测试环境搭建，开始了整个流程的运转。以下是当时的开发策略。

- **全自动地编译、部署和测试**。如图6-6所示，定时任务在每个整点触发流程，测试脚本从源代码管理系统获得最新代码，然后调用编译工具执行完整的编译。如果编译失败，测试脚本发邮件报告编译失败，然后终止流程。如果编译成功，它在一台干净的机器上部署整个系统，调用测试框架执行系统测试。最后，它汇总测试结果，生成测试报告，并用邮件发送给负责滚动测试的测试人员。
- **部署完整的系统**。对于一些分布式系统而言，将所有的子系统部署在一台机器上有些“不合情理”。但是，部署所有的子系统可以测试到所有的子系统的安装，也能够检查出由于子系统冲突造成的安装失败。安装是任何系统的核心功能，是滚动测试必不可少的测试内容。此外，在部署最新版本之前，滚动测试会反安装旧版本的系统。这也测试了系统的反安装功能。

- **用DOS脚本串联各步骤**。从以上流程可以看出，滚动测试所使用的工具无非是：操作系统的定时任务、源代码管理工具、构建工具、自动化测试框架、内部邮件系统，这些都是项目团队的常备工具。因此，用合适的方法将工具连接起来，构建出完整的流程，是架设滚动测试的主要任务。在Windows平台上，DOS Shell是最常用的脚本技术，可以满足大部分的要求，于是我们用它来连接既有工具。
- **滚动测试要能够在30分钟内完成**。如果滚动测试发现了错误，开发人员需要在滚动测试环境中调查故障原因。为了不影响下一次滚动测试，一般要留下30分钟的时间给开发人员进行诊断。

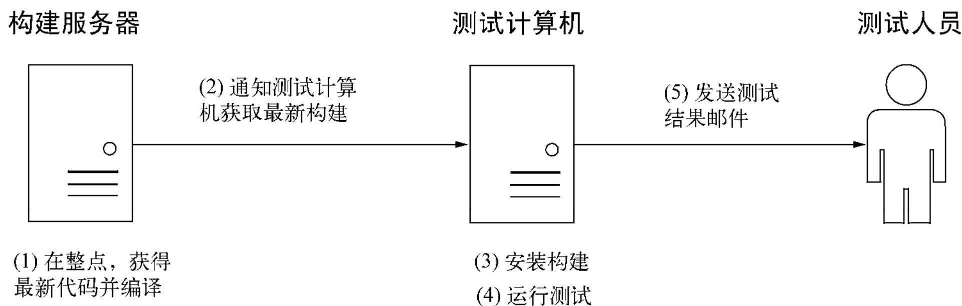


图6-6　滚动测试流程

编码完成只是测试开发的第一步，更重要的是在测试迭代中维护测试代码，让它们持续为项目作出贡献。为了快速启动滚动测试，第一版的测试用例集是较简陋的，只包含几条很简单的测试用例。随着测试的推进，我们持续改进测试用例集。以下是当时的维护策略。

- **测试用例集包含最重要的端到端系统测试**。滚动测试的主要目的是发现集成错误，因此测试执行应该能够“从输入到输出”覆盖到所有的子系统。考虑到用户价值，测试用例应该能够体现最重要的用户情景。由于滚动测试的时间有限，测试用例应该“贵精不贵多”，用较少的用户情景覆盖所有的子系统。
- **测试用例只进行必要的检查**。有些功能难以用简单的测试用例来覆盖，这时可以构建独立的测试用例集来检验它不存在严重的错误。滚动测试的目的不是检查出所有的错误，而是发现那些会阻碍进一步测试的缺陷。因此，不需要构建完备、严厉的测试用例集，只需要验证子系统的基本功能可以工作即可。
- **确保所有的测试用例是正确的**。有时系统的设计会发生变更，这会导致部分测试用例需要更新。更新测试用例的工作有可能需要几个工作日才能完成。这时，应该将这些测试用例从滚动测试中排除出去。否则，当滚动测试报告测试执行失败时，测试人员不能立即判断是被测系统的问题还是测试用例的问题，这将严重降低滚动测试的报警作用。特别是，如果测试人员“默认”测试失败是测试用例的问题，那么他有可能漏过严重的集成错误。

- **及时地维护测试用例集**。系统会增加新的模块、用户情景会发生变化、测试工具会需要更新，这都要求我们及时地维护滚动测试的测试用例集，以确保测试用例集可以覆盖到所有的子系统，且不包含错误的测试用例。
- **及时地响应测试报告**。滚动测试的威力在于根据测试结果采取及时的行动。如果对于每小时一次的错误报告采取漠视的态度，那么之前所花费的精力将毫无价值。我的做法是利用Microsot Outlook的邮件规则，对测试报告邮件生成一个通知。当测试报告到达时，Outlook会弹出一个对话框，我只需要按一下“回车”键，就可以看到报告内容。如果有问题，则进行调查；如果没有问题，按两下“取消”键就可以关闭邮件和对话框。此外，我还设置了另一条邮件规则，将所有的测试报告邮件归档到一个单独的文件夹中。

由本节讨论可以看出，计划、开发和维护是自动化测试的基本元素。测试小组需要付出持续的努力，才能让自动化测试有效地为项目服务。

6.3 计算机辅助测试

为了完成测试使命，测试人员需要开发各种类型的代码。这些代码不但可以用于执行自动化测试，还可以支持测试分析、环境配置、数据生成、失败调查、结果管理等测试任务。本章已经讨论了自动化测试，本节将介绍支持其他测试任务的软件开发。

虽然不同类型的测试开发会得到看似分离的代码，但是它们都服务于项目测试，也常常会相互调用、彼此补充，所以它们可以共享相同的开发原则和实施相似的开发方法。自动化测试的开发可以参考本节介绍的策略，测试工具的开发也可以考虑本章已经介绍的实践。

6.3.1 “交通工具”的隐喻

测试专家Johnathan Kohl提出了“交互式自动化测试”[Kohl07]，其测试开发的目标不是让自动化测试取代手工测试，而是创建测试工具来提高测试人员的能力，从而实施更有效的测试。为了更好地表达思想，他将不同的测试方式比喻为不同的“交通工具”。这是一组生动且传神的隐喻，很值得测试人员品味。本节将他的论述摘录如下。



我喜欢走着上班。我很享受风景、运动以及一边漫步一边思考问题的时光。在户外新鲜的空气中，在远离电脑的思考中，我获得了一些极好的想法。沿途的观察激发了灵感，将我引向创造性的解决方案。

但是步行是缓慢的，如果我驻足观赏可爱的动物或享受日出霞光映红的山脉，我很可能会迟到。所以，在赶时间的时候，我会搭乘公共交通。虽然公共汽车或列车仍旧伴随着步行，但是我能用快得多的速度及时地到达目的地。我极少驾车上班，因为这让我

无暇去观察和思考。但是，当别人驾车时，我能够从旅途中获得很多。例如，最近我坐列车去上班，在路上发现了以前从未注意到的河谷峭壁。我曾经多次经过河谷，但是直到坐上轻轨的那天，在没有步行或驾车分心的情况下，我才能用全新的角度观察到新的景色。

——Jonathan Kohl, Man and Machine

在Jonathan Kohl的论述中，他将不同的测试活动比喻为不同的交通方式，每种方式都有其优点与不足。

- **手工测试是步行**。在漫游产品的过程中，测试人员可以观察到丰富的细节，能够随时暂停预定的测试路线，对感兴趣的局部进行细致的测试。其不足是推进的速度较慢，有时不能在短时间内完成长距离的旅行（覆盖更多的测试目标）。
- **自动化测试是驾车**。其优点是速度快，能在短时间内测试大量的内容，其不足是自动化测试只做规定的检查，会忽视其他所有细节。即便软件出现一望即知的问题，只要测试代码没有做相应的检查，自动化测试就会放过它。其实，自动化测试更像地铁，可以将大量乘客快速地送到目的地，但是所有旅客都无法观察到地面的风景。
- **计算机辅助测试是他人驾车**。自动化测试或测试工具完成繁重的工作，带着测试人员快速地漫游产品。在测试过程中，测试人员监控产品的表现，如果发现问题，他可以“下车”调查。该策略综合了手工测试和自动化测试的优点，利用软件工具来提高测试效率、丰富测试手段，让测试人员能够更好地观察、思考与行动。

在测试中，Jonathan Kohl会使用Watri[①]、Watij[②]、JUnit[③]等程序库和框架来开发测试程序。有些测试程序完成测试准备工作，让他方便地实施手工测试。例如，测试Web应用的某个功能时，测试人员需要登录账户，并完成若干操作，才能进入目标测试页面。对此，Jonathan会编写测试程序，以驱动Web浏览器自动完成这些操作，从而将更多的时间留给测试执行。另一些程序模拟用户行为，让他可以发现手工测试不能暴露的问题。例如，他怀疑Web应用存在一个多用户并发操作的缺陷。为了模拟出多用户的情景，他编写了一个程序在一台计算机上执行用户动作，他本人则在另一台计算机上执行手工测试。在测试程序（虚拟用户）的帮助下，他发现了稳定复现该缺陷的方法。还有些程序完成手工测试难以完成的密集测试。例如，他编写了一个测试用例，向Web应用的两个字段提交超过5000个字符的字符串。该测试导致Web应用抛出异常。根据异常信息，他又增加了几个提交长字符串的测试用例，从而在几分钟内定位到缺陷的根源。

通过实践，Jonathan Kohl总结了一批计算机辅助测试的适用情景。测试人员可以参考以下测试想法，挖掘更多的测试程序需求。

---

① 用Ruby语言编写的测试Web应用的程序库，http://watir.com/。
② 用Java语言编写的测试Web应用的程序库，http://watij.com/。
③ 著名的Java语言单元测试框架，http://junit.org/。

- 用程序完成需要反复执行的测试准备工作。
    - 安装指定版本的产品。
    - 配置测试环境，如修改注册表、配置程序路径、设置产品参数、启动调试器等。
    - 驱动产品，令它完成一组操作，为后续的测试做好准备。
- 用程序产生测试数据，并让产品使用它们。
    - 生成满足特定要求的测试数据。一个典型的例子是利用PICT和测试模型生成组合测试数据（参见4.1.1节）。
    - 从其他数据源导入一批数据，做必要的清理和修改，将它们作为产品的测试数据。
    - 配置产品所依赖的数据文件或数据库，并填充测试数据。
- 用程序模拟用户操作。
    - 自动化重复操作、常用操作、典型工作流，以提高测试效率。
    - 在测试网络应用时，用测试程序模拟用户行为，以构造出多用户使用的情景。
- 用程序监控产品行为。
    - 解析产品的日志文件，报告其中的异常情况。
    - 分析产品的计算结果，报告违反业务规则的案例。
    - 监控产品的运行状态和性能指标，报告运行过程中暴露出的问题。

JonathanKohl的实践表明手工测试和自动化测试并不是相互排斥的。优秀的测试人员会综合它们的优点，开发出更具威力的测试策略，以实现单独的手工测试或自动化测试不能达到的效果。

6.3.2 选择合适的开发技术

随着开发技术的快速发展，测试人员可以选择的语言、程序库、框架、工具越来越多。他应该抱着开放的心态，学习并尝试多种的技术，了解它们的能力、优势和局限。他所掌握的技术越多，就越可能针对当前任务做出合理的设计。而且，测试任务是多种多样的，单一的技术并不适用于所有情况，这也要求测试人员灵活运用多种技术来实施高效的测试。

本节将通过一组简化的案例来展示常见的技术和方法。讨论的重点不在技术本身，而是应用技术的策略和思路。

案例1：用AutoHotKey自动化重复的输入

AutoHotKey[①]是一个开源的鼠标与键盘宏工具。它提供了一个简单的脚本语言，让用户编写

① http://www.autohotkey.com/。

脚本来定义快捷键和对应的宏命令。在加载了用户脚本之后，它在后台监听用户的键盘输入。一旦用户输入预先定义的的快捷键，它就会执行宏命令，以自动完成一系列操作。

在手工测试中，测试人员有时得重复执行一组机械操作。为了提高测试效率，他可以编写AutoHotKey脚本来自动化此类重复操作。

例如，测试人员测试PowerPoint的图片导出功能，测试任务需要检查导出的PNG图片。他的测试策略是制作多种不同类型的幻灯片页面，然后输出为PNG图片，并审阅输出结果。为了将文档导出为PNG图片，他可以点击“File Export Change File Type PNG Portable Network Graphics (.*png) Save As”（详见图6-7），以启动“另存为”对话框，然后输入目标路径和文件名。他发现这是一组会反复执行的机械操作，于是编写了代码清单6-5所示的AutoHotKey脚本。该脚本的作用是，当测试人员按下Ctrl+F12，AutoHotKey就会发送键盘消息，驱动PowerPoint弹出“另存为”对话框，并将保存路径设为指定值。之后，测试人员只要输入文件名，就可以完成图片导出操作。

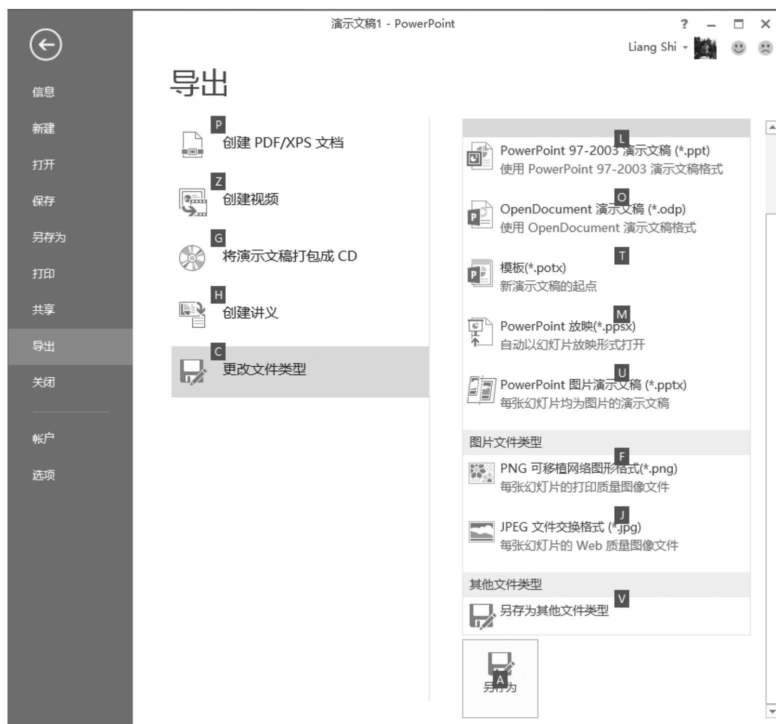


图6-7　PowerPoint的导出功能

代码清单6-5　执行导出图片操作的AutoHotKey脚本

```
^F12::              ; 快捷键是Ctrl+F12
Send !F             ; 发送"Alt+F"以选择"File"
Send E              ; 选择"Export"
```

```
Send C              ; 选择"Change File Type"
Send F              ; 选择"PNG Portable Network Graphics (*.png)"
Send A              ; 选择"Save As"
WinWait, Save As    ; 等待"Save As"对话框出现
Send C:\temp\       ; 发送目标路径
Send {Enter}        ; 发送"回车"，令对话框的当前路径转移到目标路径（C:\temp）
return              ; 结束
```

对于了解AutoHotKey的测试人员，编写代码清单6-5只要花费几分钟的时间，其成果是用快捷键取代了一组键盘操作。这看似是微不足道的进步，但这有利于测试人员摆脱枯燥的重复动作，将精力集中在高智力挑战的任务上。实际上，测试过程蕴含许多通过自动化来提高操作效率的情景。如果测试人员积极地识别自动化机会，并合理利用，他可以积累一系列小的自动化成果来显著提高测试效率。

AutoHotKey并不是完成此类任务的唯一工具①，测试人员需要根据产品类型和当前任务选择合适的工具。例如，Jonathan Kohl对于Web应用选择了Watri和Watij来控制浏览器和页面。使用恰当的工具，测试人员只需编写很短小的代码就可以达成目标。使用不恰当的工具，测试人员会耗费大量时间于测试代码的编写和调试，以至于压缩了测试产品的时间。

案例2：用DOS脚本自动化测试环境配置

案例1是一个用计算机加速测试执行的例子，本案例是利用脚本来进行测试准备。我经常创建虚拟机来执行测试。在获得一个新的虚拟机之后，我会用管理员权限运行代码清单6-6所示的DOS脚本②。

代码清单6-6　配置测试环境的DOS脚本

```
REM 1) setup variables ---------------------------------------------
set src=\\myserver\setup
set tgt=c:
set tools=%tgt%\tools

REM 2) setup debuggers ---------------------------------------------
xcopy %src%\debuggers   %tgt%\debuggers\    /Y /E /H

md c:\symbols
%tgt%\debuggers\windbg.exe -IS

REM 3) setup kernel debugging --------------------------------------
bcdedit /debug on
bcdedit /dbgsettings serial debugport:1 baudrate:115200

REM 4) copy tools --------------------------------------------------
xcopy %src%\tools %tools%\    /Y /E /H

REM 5) config machine ----------------------------------------------
reg.exe import %tools%\reg\powershell_config.reg
```

① 对于PowerPoint而言，实现快速图片导出的另一种方法是执行VBA语句：

```
Application.Presentations.Item(1).SaveAs "c:\temp\test", ppSaveAsPNG, msoFalse
```

② 代码清单6-6是一个简化的脚本，真实的脚本要更复杂一些。

```
reg.exe import %tools%\reg\windbg_config.reg
reg.exe import %tools%\reg\console_config.reg

setx PATH "%PATH%;%tgt%\debuggers;%tools%"

REM 6) map network driver -------------------------------------------
net use z: \\myserver\share

REM 7) setup fiddler2 -----------------------------------------------
start %tools%\install\Fiddler4BetaSetup.exe
```

代码清单6-6将一批工具从服务器复制到当前计算机，安装软件，并修改操作系统的设置。各步骤的概要解释如下。

(1) 建立变量，设定文件要从何处获取、要复制到本机的哪个目录。

(2) 复制Windbg到本机，并将它设定为事后调试器。

(3) 开启本机的内核调试。

(4) 复制一组文件和工具到本机。

(5) 导入注册表文件来修改计算机配置。将Windbg的目录和工具目录加入到环境变量PATH。

(6) 将共享的网络路径映射为本机的Z盘。

(7) 安装HTTP调试代理Fiddler2。

案例2说明利用操作系统的Shell和内建命令，测试人员可以快速地配置操作系统和软件工具，获得准备就绪的测试环境。在许多情景中，测试人员执行的任务与系统管理员很类似，都需要配置软硬件参数、安装软件和维护环境。学习一些系统管理员使用的脚本和工具，能够帮助测试人员更高效地完成测试准备。

案例3：用PowerShell和Excel分析产品性能

测试过程会产生许多信息，有些信息需要在测试执行之后做进一步的分析。例如，测试人员需要分析3个重要操作（FetchData、ExtractData、ComputeData）的性能。于是，他用测试代码将这些操作的用时写入测试日志，具体实现请参考代码清单6-7。

代码清单6-7　将操作的用时写入日志文件

```
// 测试辅助类：利用C#的Dispose记录操作的开始、结束和用时
public class TimeLogger : IDisposable
{
    DateTime statTime;
    string action;

    public TimeLogger(string action)
    {
        // 记录操作开始
        TestLog.Log("BeginAction\t{0}", actionMessage);
        this.statTime = System.DateTime.Now;
        this.actionMessage = action;
    }
```

```
    public void Dispose()
    {
        // 记录操作结束
        // 记录格式: "EndAction<Tab>[操作名]<Tab>[用时] seconds"
        TestLog.Log("EndAction\t{0}\t{1} seconds",
            this.action,
            (System.DateTime.Now - this.statTime).TotalSeconds);
    }
}

...

// 记录操作FetchData的用时
using (var logger = new TimeLogger("FetchData"))
{
    // FetchData的具体操作
}

// 记录操作ExtractData的用时
using (var logger = new TimeLogger("ExtractData"))
{
    // ExtractData的具体操作
}

// 记录操作ComputeData的用时
using (var logger = new TimeLogger("ComputeData"))
{
    // ComputeData的具体操作
}
```

在测试执行之后，测试人员使用代码清单6-8的PowerShell脚本来搜索日志目录下的日志文件，然后用正则表达式“EndAction\t(.?)\t(.?)\sseconds”来匹配记录操作用时的文本行。如果匹配成功，脚本提取操作名和用时，将它们输出到perf.txt。图6-8展示了该脚本的输出结果。

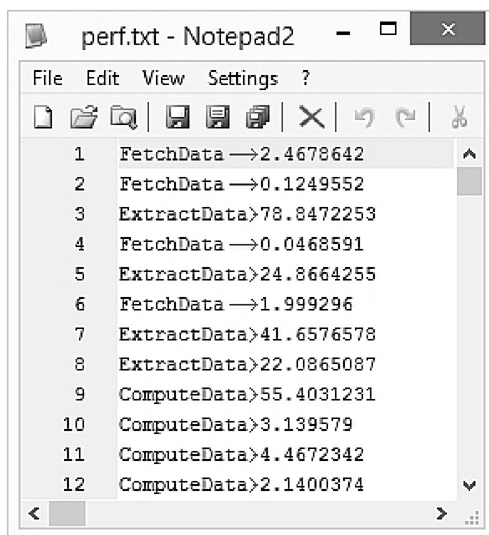


图6-8 分析测试日志所获得的操作和用时

代码清单6-8　用正则表达式分析测试日志，获得操作的用时

```
Get-ChildItem -Filter *.log -Recurse |
    Select-String "EndAction\t(.*?)\t(.*?)\sseconds" |
    Select -ExpandProperty Matches |
    foreach { '{0}`t{1}' -f $_.Groups[1].Value, $_.Groups[2].Value } > perf.txt
```

然后，测试人员用Excel导入perf.txt，获得如图6-9所示的数据表。基于该数据表，测试人员生成如图6-10所示的透视图，以分析各个操作的平均用时。通过该图，测试人员可以直观地了解到FetchData的速度最快（平均用时为0.45秒）、ExtractData的速度最慢（平均用时20.45秒）、ComputerData的速度接近ExtractData（平均用时为16.15秒）。

Action	Time
FetchData	2.4678642
FetchData	0.1249552
ExtractData	78.8472253
FetchData	0.0468591
ExtractData	24.8664255
FetchData	1.999296
ExtractData	41.6576578
ExtractData	22.0865087
ComputeData	55.4031231
ComputeData	3.139579
ComputeData	4.4672342
ComputeData	2.1400374

图6-9　包含操作和用时的Excel数据表（局部）

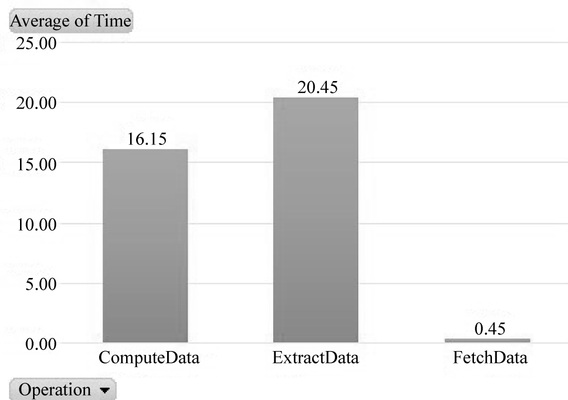


图6-10　反映操作平均用时的透视图

为了进一步了解情况，他又在透视图中加入各个操作的最长用时，获得如图6-11所示的结果。这时，他惊讶地发现ExtractData的最长用时达到432.32秒，是平均用时的20多倍。于是，他生成ExtractData各次用时的散点图（如图6-12所示）。由该图可知，测试过程执行ExtractData达4000多

次，该操作的用时大多小于50秒，只有一次超乎寻常地达到432.32秒。该图不但很好地反映了ExtractData的用时情况，还建议测试人员应该调查那次超长的用时，发掘它背后隐藏的问题。

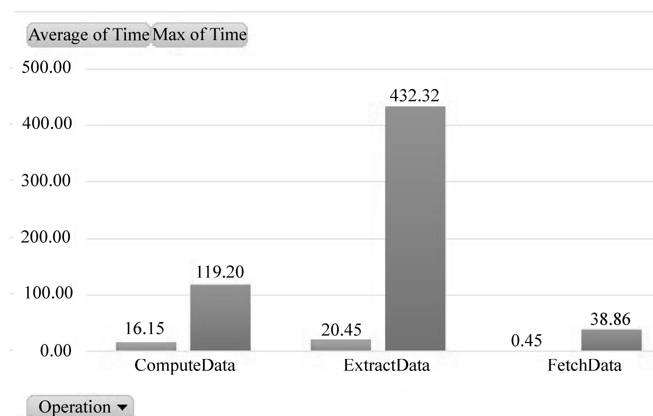


图6-11　操作的平均用时和最长用时

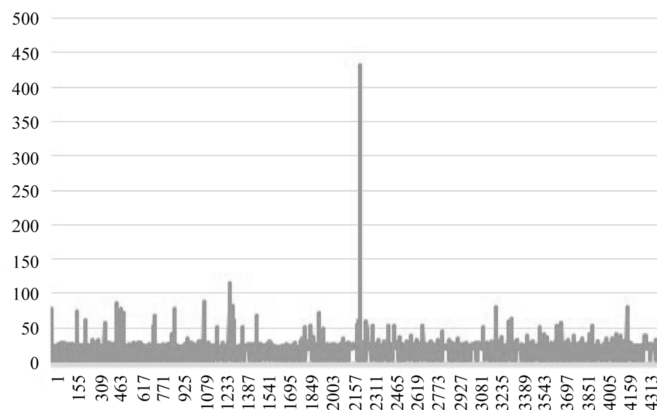


图6-12　ExtractData用时的散点图

回顾以上测试开发与分析过程，不难获得以下经验。

- 如果测试代码用清晰的格式记录测试执行的信息，测试日志将为成为后续分析的重要数据源。
- 利用动态语言和正则表达式，可以编写短小的脚本来提取值得分析的数据。
- 电子表格软件是方便的数据可视化工具。利用它，测试人员无需编写代码，就可以生成多种类型的图表。
- 电子表格软件也是探索数据、挖掘知识的工具。测试人员根据已有的图表来提出疑问，然后设计出新的图表来提供答案。这让测试人员从多个角度考察数据，从而建立对产品特性的完整理解。

❑ 测试开发可以借助多个工具，通过组合它们的长处来快速完成测试任务。

案例4：用IronPython分析系统负载

我曾经测试过一个作业处理系统，它的任务是计算用户每天提交的作业。为了避免当天的作业积压到第二天，项目团队利用计算机集群搭建了6条作业流水线，使该系统能够同时处理6个作业。为了分析产品环境和测试环境中系统的性能，我用IronPython[1]编写了一个脚本perf.py来绘制系统对流水线的利用率。该脚本的运行步骤与案例3的数据处理流程很相似。

(1) **提取数据**：perf.py从系统日志中分析出每个作业的类型、开始时间和结束时间。该日志可能来自产品环境，也可能来自性能测试环境。

(2) **处理数据**：perf.py的目标是在一个窗口中绘制出作业的运行情况。为此，它将窗口视为一个画布，将6条流水线视为画布上等高的6行，将一个作业视为某行中的一个矩形。它根据作业开始时间和结束时间，计算出对应矩形的长度、宽度和左上角的坐标。

(3) **可视化数据**：perf.py根据矩形的长度、宽度和左上角的左边，在窗口中绘制出所有矩形[2]。图6-13展示了最终的绘制结果。图的横轴对应着时间，纵轴对应着流水线，每一个矩形对应一个作业，矩形越宽说明作业处理时间越长，相同颜色的作业属于同一种作业类型。

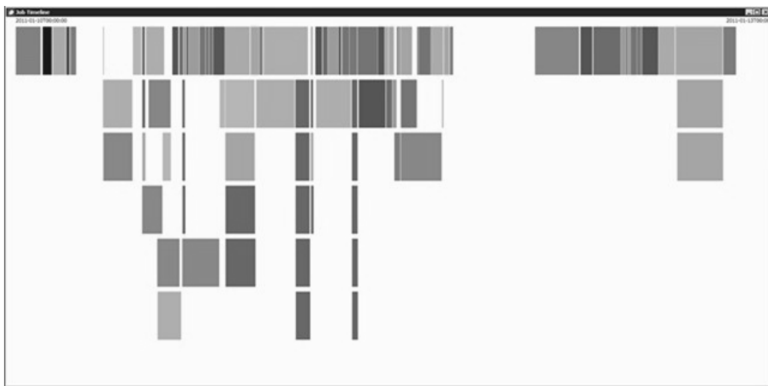


图6-13　用图形来展示系统的负载

根据图6-13，我可以直观地分析数据，了解在所分析的时段内，系统完成了哪些作业，何种作业用时最长，何种作业目数目最多，哪些作业在并发执行，哪些作业在串行执行，哪些时段系统繁忙，哪些时段系统空闲。基于这些信息，我可以理解产品环境的作业运行特点，设计更真实的性能测试方案，还可以分析出系统的性能瓶颈，发现值得性能优化的作业类型。此外，perf.py

① IronPython是Python语言在.NET平台上的实现，主页：http://ironpython.codeplex.com。

② 该脚本调用的绘图API是WPF (Windows Presentation Foundation)，主页：http://msdn.microsoft.com/en-us/library/ms754130.aspx。

还可以分析性能测试环境，帮助我发现性能测试的不足，进而改进测试方案，以提供更接近产品环境的工作负载。

在.NET程序库和我日常积累的程序库帮助下，perf.py仅用300余行就完成了提取数据、处理数据、可视化数据的任务。代码清单6-9摘录了脚本的核心逻辑。由这段代码，不难看出IronPython的一些优点。

- Python语言简洁清晰，表达力强，能用比主流静态语言（如Java、C#等）更短的程序完成计算任务。
- IronPython可以直接调用包括WPF在内的.NET程序库，这降低了数据可视化的工作量。此外，IronPython还可以调用CPython程序库，并通过.NET互操作技术调用COM对象和Win32 API。这些都显著提高了开发效率。

代码清单6-9　计算矩形位置和绘制矩形的核心代码

```
# 计算矩形位置的类
class ArrangeRectangle:
    # ...

    # 将矩形（作业）放入恰当的行（流水线）
    def compute_position(self):
        rows = [[] for i in range(pipeline_count)]
        for r in self.rectangles:
            index = self.get_row_index(rows, r)
            rows[index].append(r)
            r.top = index * (self.row_heigth + row_gap)

    # 寻找可以容纳矩形的行
    def get_row_index(self, rows, rectangle):
        for index, row in enumerate(rows):
            if self.has_space(row, rectangle):
                return index
        rows.append([])
        return len(rows) - 1

    # 判断行row能否容纳矩形rectangle
    def has_space(self, row, rectangle):
        for r in row:
            if (self.is_conflict(r, rectangle)
                or self.is_conflict(rectangle, r)):
                return False
        return True

    # 判断两个矩形（rx和ry）的位置是否冲突
    def is_conflict(self, rx, ry):
        if ry.start <= rx.start < ry.end:
            return True
        elif ry.start <= rx.start and rx.end <= ry.end:
            return True
        else:
```

```
            return False

    # 绘制矩形的类
    class DrawRectangle:
        # ...

        # 绘制所有的矩形，并将它们加入画布
        def add_rectangleangles(self, rectangles):
            c = self.canvas
            for r in rectangles:
                wr = self.build_wpf_rectangle(r)
                self.add(wr, r.left, r.top)

        # 绘制一个矩形
        def build_wpf_rectangle(self, rectangle):
            wr = wpf.Rectangle()
            wr.Width = rectangle.width
            wr.Height = rectangle.height
            wr.Fill = self.get_color(rectangle.type)

            tt = wpf.ToolTip()
            tt.Content = rectangle.text
            wr.ToolTip = tt
            return wr
```

动态语言的另一个优势是部署和修改方便。因为我所使用的测试计算机都安装了.NET Framework，所以我只需要复制IronPython解释器（大约4MB）和几个脚本文件就完成了应用部署。这方便我在不同的环境（预发布环境、集成测试环境、性能测试环境、功能测试环境等）下运行脚本，获得系统的性能信息。在数据分析过程中，我可能会产生一些新的数据可视化想法。这时，我可以用文本编辑器修改脚本，然后运行脚本来获得直观的反馈，从而激发出更多的数据分析思路。因为许多测试环境不包含大型的开发工具，所以它们不支持主流静态语言的快速开发。动态语言无需编译、立即可用的特点让我可以灵活快速地修改脚本，以满足随时浮现的测试需求。

由本节的讨论可知，测试任务中存在许多运用程序来提高工作效率的机会。测试人员需要抓住这些机会，综合运用多种开发技术来构建不同类型的测试工具。

6.4　大规模自动化测试

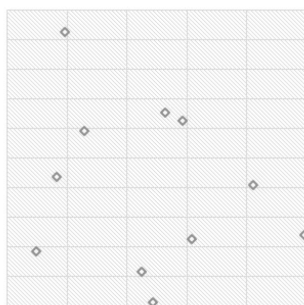
大规模自动化测试（high volume automated testing，HiVAT）是一种测试策略，通过自动地执行海量的测试用例，来强力检测产品缺陷[Kaner13]。随着硬件性能的提高和集群计算技术的发展，HiVAT已成为一种威力巨大的自动化测试策略。本节讨论它的基本概念和设计因素。

6.4.1　基本概念

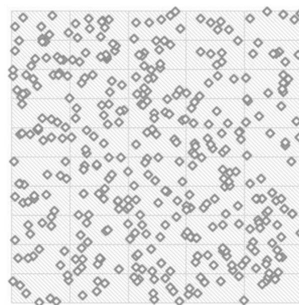
许多常见测试技术的出发点是获得数量较少、有代表性的测试用例，通过少量的典型采样来

评估整个测试空间。例如等价类划分将测试输入划分为若干个等价类，从中选择测试用例。又例如组合测试根据测试模型，生成符合组合覆盖标准的最小测试用例集。

HiVAT的战术恰恰与这些技术相反。它所运行的每一个测试都是平凡的，但是海量的测试用例密集地覆盖了测试空间，能够发现精心构造的小规模测试用例集所错过的缺陷。图6-14对比了这两种测试策略，其中每一个点都是一个测试用例，包含点的方形区域是测试空间。左侧的测试技术生成少数的测试用例，对测试空间的采样较稀疏；右侧的测试技术则产生大量的测试用例，对测试空间的采样较密集。



基于最佳采样的测试技术（组合测试等）



基于大规模采样的测试技术（HiVAT）

图6-14 基于不同采样策略的测试技术

HiVAT常用于测试复杂的功能、流程或产品。此时，小规模的测试用例集不能提供足够的测试信心。测试人员需要借助计算机的能力来运行大规模的测试，以发现隐蔽的错误。以下是一些HiVAT的典型案例。

- **5.3.3节介绍了Harry Robinson测试地图导航的方法**。测试程序利用美国所有邮政编码的两两组合作为测试输入（大约十亿条测试用例），用一组约束规则来检查所获得的导航路线集。如果规则检查发现了失败的案例，测试程序会将该案例报告给Harry，请他做进一步的分析。
- **Mozilla项目小组需要检查Firefox在加载页面时是否出现内存泄漏、断言触发和程序崩溃等问题**。他们的测试策略是从alexa.com获得最流行的一百万个URL，用它们来检测Firefox的健壮性[Johnson10]。在测试环境中，测试程序启动指定版本的Firefox，令它打开一个URL，然后加载Firefox的一个测试插件，该插件会遍历页面上的链接，并执行一组检查。在打开URL、遍历链接和执行检查的过程中，测试程序会详细记录所发现的问题，并保存导致问题的页面。测试一百万个URL需要一个月左右的时间。为了加快测试反馈的速度，Mozilla小组还构建了一个包含一千条URL的测试集合，用于每日测试。
- **模糊测试是一种修改输入数据来暴露软件缺陷的方法**。如果某款产品需要读取复杂的文档，测试小组可以使用文件模糊器，对原始文档进行随机修改，生成大量的模糊文档。然后，模糊测试框架启动产品，令它读取这批被篡改的文档，并严密监控可能的异常情

况。在测试结束后，测试人员分析测试日志中记录的错误，以识别安全缺陷。模糊测试是一种暴力测试方法，一轮模糊测试会使用数万个甚至更多的模糊文件。如果运用得当，模糊测试可以发现许多手工测试或简单的自动化测试无法发现的问题，这对于提高软件的安全性极具价值。例如Microsoft Office团队在开发Office 2013时，通过模糊测试发现并修改了2100多个缺陷[Gallagher12]。

- **猴子测试是利用猴子程序来随机地测试产品**。猴子程序是一个不知疲倦的测试程序，它随机地从动作列表中选择一个动作来操作产品，并监视产品的状态和行为，记录所发现的问题。猴子程序会连续运行整晚或几天，如果产品崩溃或停滞，它会重启产品以继续测试。对于简单的猴子程序，动作列表是一组键盘、鼠标、触控命令，它会纯粹地随机选择命令，而不考虑产品当前的状态。这类猴子实现简单，可以快速投入测试，但是只能发现程序崩溃、程序停滞、资源泄漏等常见错误。对于智能的猴子程序，其动作列表是一组业务操作（有时是简单的功能测试），它会根据产品的状态来选择业务操作，并检查操作结束后软件的状态是否正确。智能的猴子拥有一些产品的知识，所以它的命令和检查更有针对性，不但能够发现常见错误，还可以挖掘出产品特有的缺陷。

HiVAT拥有悠久的历史，Cem Kaner在1980年就发现了一些测试团队运行大量测试用例来探测复杂软件的深层次缺陷。近年来，随着软硬件技术的发展，HiVAT的有效性和重要性与日俱增。第一，软件产品越来越复杂，少量的测试用例不能提供足够的测试覆盖，测试人员需要更好的办法来覆盖产品的代码和状态。第二，计算机硬件的价格在持续下降，性能在稳步提升，这为运行大批量的测试用例提供了硬件基础。第三，虚拟化技术让一台物理计算机运行多台虚拟计算机，这使得项目团队能够更充分地利用空闲计算资源。第四，云计算技术让测试人员可以用多台（虚拟）计算机同时运行海量的测试用例，并汇总测试结果，这极大地提高了测试运行的速度。例如，Harry Robinson在测试地图导航时运行了上亿条测试，这在以往的测试环境中是不可想象的。但是，利用云计算技术，他可以使用上千台计算机同时执行测试，能够在几天内完成数量如此庞大的测试。

可见，新的测试挑战和测试平台要求测试人员重新思考自动化测试的使命，用创新思维去充分利用计算资源，以提出更具威力的测试策略。测试人员需要思考：如果我拥有10台计算机来运行测试用例，我应该设计何种自动化测试？如果有100台计算机，又怎么设计？如果有1000台计算机，又如何设计？这些问题看似有些遥远，其实是很多测试人员需要考虑的真实情况。目前，许多测试小组拥有测试实验室，一般配有几十台计算机，某些项目的测试环境甚至拥有上千台计算机。在很多时候，这些计算机并没有满负荷运转，许多计算资源都被闲置。测试人员应该考虑如何充分利用这些空闲资源，从而大幅度提高测试的广度和深度。

6.4.2　测试设计

HiVAT有多种实施方式，也存在大量创新空间，然而其基本流程具备一定的模式。图6-15是

利用多台计算机进行HiVAT的典型工作方式，其流程是将测试任务分发给多台计算机，然后汇总来自所有计算机的测试结果。以下是基本步骤的描述[①]。

(1) 测试人员使用工具生成大量的测试用例。

(2) 根据可用计算机的数量，测试控制服务器把测试用例划分为若干组，将它们分配给测试执行计算机。

(3) 在测试执行计算机上，测试程序运行测试用例，并记录测试日志。

(4) 测试控制服务器汇总来自各台计算机的测试结果，将它们保存到测试结果仓库中。

(5) 测试人员调查测试所发现的问题，并提交缺陷报告。

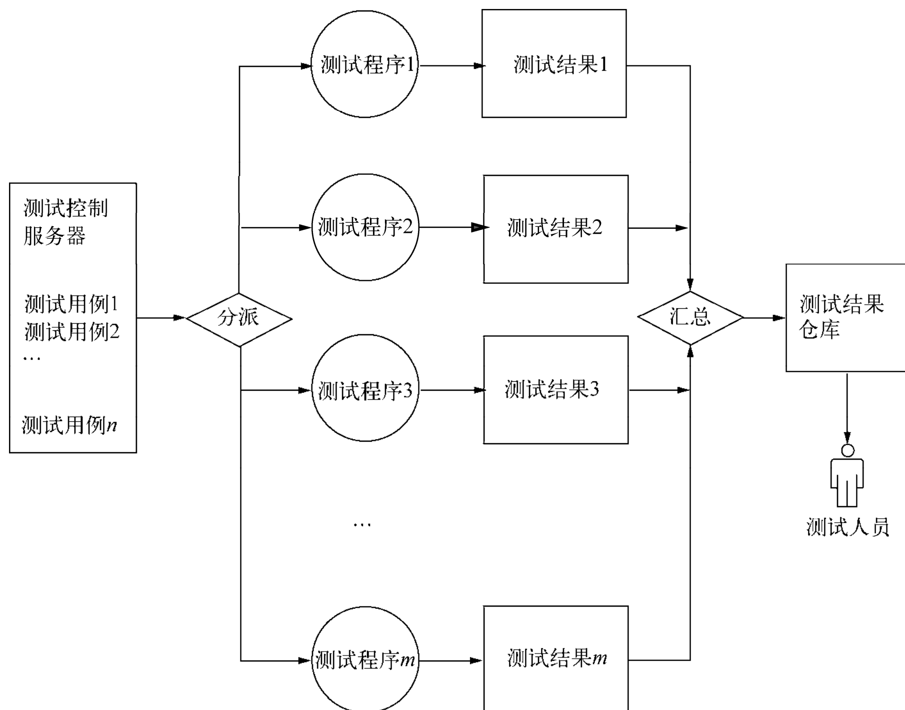


图6-15　用多台计算机执行HiVAT的一种架构

因为HiVAT会运行海量的测试用例，测试过程会面临一些特殊的挑战，测试人员需要根据产品的特征和所依赖的自动化技术做出恰当的设计。本节针对HiVAT的特点，讨论其设计与实现过程中需要重点考虑的问题。

问题1：如何生成大量的测试输入数据

HiVAT的威力来自于执行海量的测试用例，测试输入数据的差异性和代表性将极大地影响测

① 对于在一台计算机上进行的HiVAT，步骤(2)和步骤(4)可以省略。

试效果。优质的测试输入使投入的计算资源获得充分的回报，不良的测试输入则浪费了庞大的计算资源和漫长的测试时间。以下是一些构建测试输入数据的常见方法。

- **穷举输入域的所有取值**。例如，Harry Robinson穷举了美国邮政编码的两两组合，获得地图导航的测试输入数据。又例如，测试人员在测试拥有多个输入变量的模块时，不使用两因素测试用例集，而使用全组合测试用例集。
- **尽可能多地收集真实的业务数据**。例如，Mozilla项目小组从alexa.com获得了最流行的一百万个URL，用来测试Firefox的页面加载。如果测试对象是产品的文档读取功能，测试人员可以从各种途径收集不同用户的真实文档。如果测试对象是Web应用，测试人员可以分析产品日志，获得一段时间内用户群的操作和输入数据，从而构建出测试输入集。
- **随机地产生测试数据**。例如，模糊测试调用文件模糊器随机地修改原始文件，生成一批测试文件。因为对文件的修改方式是无穷尽的，穷举法在此处不适用。大规模随机修改可以使产生的文件较均匀地覆盖输入空间，提高发现缺陷的概率。在数学上，数量庞大的随机采样有助于均匀地探测整个测试空间，降低遗漏重要输入区域的可能性。
- **用启发式规则指导随机采样**。纯粹的随机采样不考虑产品的业务知识，虽然确保了数学性质上的“公平”与“均匀”，但是没有根据产品的特点在重点区域进行密集测试。为了提高测试效率，测试人员可以用一些启发式规则来指导测试数据生成。例如，高级的文件模糊器会读取文档格式的规格说明，以建议文档格式的模型，然后根据该模型来修改文件。其修改过程依旧具备很强的随机性，但是可以生成更有攻击性的一批文件。又例如，测试人员根据业务知识将输入域划分为若干个区域，分别赋予不同的权重。测试数据生成器会在高权重的区域多采样，在低权重的区域少采样，以便重点测试高权重的区域。
- **为每一条测试数据生成唯一的编号**。在测试过程中，测试程序在测试日志中记录测试数据编号和对应的测试结果。当测试人员分析测试日志时，他可以根据编号找出测试数据，以复现相应的测试结果。这对于测试数据和测试结果的追踪管理尤其重要。

以上测试数据生成方法并不相互排斥。当测试复杂产品时，测试人员需要综合运用它们，以发挥各自的优点。例如，测试人员收集一大批用户文档来测试产品的文档读写功能（功能性测试）。然后，他用文件模糊器基于这批用户文档生成一组模糊文件，以检查文档读写功能是否存在安全缺陷（安全性测试）。接着，他针对文档中一个特别重要的字段，配置文件模糊器去生成一批文档，以实现对该字段的穷举测试。

HiVAT往往会生成大量的测试输入数据，随之而来的一组问题是：已生成的测试数据是否已经足够好？是否需要生成新的测试数据？新的测试数据需要具备哪些新的特征？对于非穷举测试，这些问题没有标准答案。

通常，测试人员可以从两个角度来考察测试数据集的质量。第一，测试数据集应该很好地覆盖了业务领域。例如，地图导航测试应该覆盖了所有典型的出发地和目的地，页面加载测试应该

覆盖了大部分用户可能访问的页面类型，模糊测试应该覆盖了文档格式的所有字段。在理想情况下，测试输入应该具备很好的典型性和差异性，既能覆盖所有业务数据类型，又能提供足够多样化的细节。第二，测试数据集应该很好地覆盖了代码实现。测试人员可以用代码覆盖率、状态覆盖率等方法来寻找尚未被充分测试的代码，然后用这些信息来指导后续测试数据的生成。

问题2：如何构建测试先知，以自动地发现产品缺陷

HiVAT会执行大量的测试，手工地检查所有的测试结果是不现实的，因此需要测试先知来自动地识别缺陷或记录值得调查的问题。与5.3节介绍的测试先知技术相似，一致性检查和规则检查是自动化测试先知的常见构造方法。以下是一些常见策略。

- **检查产品和参考程序的一致性**。例如，Open Office Calc是一款电子表格软件[①]。为了测试它的公式计算，测试程序将一组包含公式的电子表格文档作为测试输入，让Calc打开这批文档，并获得它的公式计算结果。然后，测试程序将这些结果与Microsoft Excel的计算结果进行比较，识别出不一致的结果值。将Excel作为参考程序简化了测试开发，让测试程序可以立即投入应用。虽然Excel也可能存在缺陷，但是从头实现相同质量的公式计算程序可能是测试小组无法承担的任务。
- **检查被测版本和已发布版本的一致性**。例如，某绘图软件需要支持已发布版本所制作的文档。为此，测试程序将一组真实的用户文档作为测试输入，让被测版本打开这批文档，并通过屏幕截图来获取它所绘制出的图形。然后，测试程序将被测版本的截图与已发布版本的截图进行比较，识别出不一致的绘制结果。
- **检查产品自身的行为是否一致**。例如，Harry Robinson用一组规则来检查谷歌地图给出的导航路线。其中一条规则是“从A到B的导航路线的长度与从B到A的导航路线的长度没有显著差别”。该规则要求导航计算在两个方向上保持一致，即检查产品的逆反运算是否等价。
- **用一组约束规则来检查产品的结果**。例如，Mozilla项目小组测试Firefox页面加载时，他们实现的测试程序会对Firefox做一系列检查，包括检查Firefox是否泄漏了内存，检查Firefox是否触发了断言，检查Firefox在加载URL和页面链接时是否崩溃等。这些检查可以视为一组规则，虽然不能发现所有缺陷，但是对于特定类型的问题有很强的检测能力。
- **基于测试模型的检查**。有些测试程序建立了产品的状态机模型，并用它检查产品的行为。在测试时，测试程序获取产品的当前状态，随机地选择一条命令发送给产品，并检查它的反应。如果命令可以触发符合状态机模型的变迁，产品应该迁移到目标状态，并执行相应的动作。如果命令不能触发合法的变迁，产品应该拒绝执行。利用状态机模型，测试程序可以用漫长的命令序列来检查软件的行为，并发现一些长期执行才会暴露的问题。

在测试实践中，测试程序往往会实现多种测试先知，从不同的角度来识别缺陷。例如，测试程序可以用状态机模型来驱动产品的状态变迁，并记录异常的变迁。与此同时，它会检查产品是

① http://www.openoffice.org/product/calc.html。

否存在资源泄露、断言触发和程序崩溃等问题，并用参考程序来检验产品的计算结果。只要计算资源和测试时间允许，多样化的测试先知将发现更多的问题，有助于提高HiVAT的产出。

问题3：如何监控测试执行

因为HiVAT运行时间漫长，会触发一些难以预料的产品错误，测试平台需要一些功能来保证大批量的测试可以执行完毕。以下是一些常见的测试设计。

- **用独立的监控进程监视测试程序和被测产品**。在测试过程中，测试程序和被测产品都可能出现故障，陷入停滞状态或发生崩溃。为了让测试可以持续运转，测试人员可以实现一个简单的监控进程，用它观察测试程序和产品的状态。如果发现某个进程陷入停滞，它会终止该进程，再重新启动它。如果发现某个进程崩溃，它会收集相应的内存转储文件，并重启该进程。
- **测试程序记录测试进度，以便重新启动后继续运行尚未执行的测试用例**。例如，测试程序获得了1000条测试用例，每条测试用例有唯一的编号。它先用测试日志记录所有的测试用例，然后开始测试。在测试过程中，它执行完一条测试用例，就记录该测试用例的编号。如果测试程序在执行第401个测试用例时发生故障，被监控进程终止。重新启动后，测试程序会分析测试日志，然后从第402个测试用例开始执行。这避免了重复执行测试用例的开销，也保证测试程序不会遗漏测试用例。
- **测试程序主动向测试控制服务器申请测试用例，即使用拉模式来分配测试用例**。在使用多台计算机执行测试时，有一些计算机的性能比较高，能够更快地完成测试用例。拉模式允许这些计算机在执行完一批测试用例之后，可以立即获得新的一批测试用例，而不必等待哪些速度较慢的计算机。这有助于提高计算机集群的整体性能。
- **当某台计算机发生故障时，测试控制服务器应该记录错误，并将分配给它的测试用例集转移到其他计算机**。因为软件、操作系统、硬件都可能发生故障，所以测试执行计算机停止运转是需要考虑因素。分配测试用例的控制服务器应该记录哪些计算机获得了哪些测试用例，并周期性地联系测试执行计算机，以获取测试进度。当某台计算机停止响应时，服务器应该将它尚未执行的测试用例放回总测试用例集。这样，当其他计算机申请测试用例时，这些测试用例可以被再次分配。

HiVAT需要在无人职守的情况下长时间运行。测试人员需要考虑测试平台的容错性和健壮性，并做出一些相应的设计。紧密的监控和恰当的故障恢复是基本的考虑点。

问题4：如何调查大量的测试失败

HiVAT会产生大量的测试结果。单纯的手工调查费时费力，让测试人员成为测试流程的瓶颈。而且调查所用的时间越长，测试人员越容易感到疲惫，越可能漏掉一些产品缺陷。为了帮助测试人员高效地评估测试结果和调查失败，测试平台需要提供相应的自动化支持。以下是一些常见的测试设计。

- **测试程序应该记录它执行的所有测试用例或命令，以便测试人员重现缺陷**。在猴子测试等长时间运行的随机测试中，测试程序应该用固定的格式记录所有发给产品的命令。测试人员可以根据测试日志，手动执行命令序列，也可以用工具读取测试日志，来重新发送命令序列。这对于重现由随机命令序列发现的软件缺陷很有帮助。
- **一旦测试程序发现软件错误，它应该尽可能多地保存产品信息和环境信息**。例如，测试程序可以保存崩溃软件的内存转储文件，并在测试日志中记录该转储文件的路径。又例如，测试程序发现导航路线有误时，它可以记录起点和终点的位置信息（邮政编码），并保存导航路线的网页截图，帮助测试人员立即了解失败情况。又例如，测试程序发现浏览器在打开某个URL时崩溃，它除了保存内存转储文件，还会将该URL保存为离线网页。这使失败调查无需在网络传输上耗费时间，也避免在线网页的变化使得缺陷不能重现。
- **用多台计算机执行测试时，测试程序应该将测试结果提交到一个中心节点**。这让测试人员可以集中审阅和管理所有测试结果，避免了手工收集多份结果的开销。通常，中心节点是一个基于数据库的Web服务，测试程序调用它的接口来提交测试结果。该服务会整理测试结果，并将它们保存在数据库中。另一种设计是编写一个数据收集程序，它扫描所有的测试执行计算机来收集测试结果，并将整理后的数据保存在数据库中。
- **让测试人员可以通过查询命令来获取特定的测试结果**。为了支持测试调查，测试平台在显示测试结果时需要支持排序、过滤、分组等功能。将测试结果保存在关系型数据库是支持此需求的典型设计。测试人员可以通过SQL查询来分析每个测试失败的错误消息，从而分组相似的失败、过滤已知的失败、查询某个失败的出现频率等。此外，数据库能够保存以往的测试结果，这方便测试人员比较当前版本和历史版本的结果，从而快速发现新的失败。

恰如6.2.3节所讨论的，测试人员应该开发面向调试的测试代码，该实践在HiVAT中更加重要。在设计之初，测试人员就应该考虑如何分析HiVAT产生的海量测试结果，并在测试生成、测试执行、测试报告等各个环节做出相应的设计。实际运行HiVAT后，测试人员会发现测试调查面临一些新的挑战。这时，他需要调整设计，使HiVAT的调查效率随着测试迭代而逐步提高。

6.5　小结

本章讨论了测试开发的基本分类，介绍了一些自动化测试和计算机辅助测试的目标、设计、实现和经验。

- 测试开发的对象包括自动执行的测试用例和辅助测试活动的工具。
- 好的测试开发策略的4个特征：产品相关、聚焦风险、多种多样、讲求实用。
- 测试开发也是一种软件开发，拥有规划、设计、实现、运维等活动，需要项目管理、测试设计、代码实现等技能。

- 自动化测试金字塔是一个概念模型，描述了自动化测试用例在理想情况下的分布。测试人员应该通过逐步交付有价值的测试用例来达成该目标。
- 可调试性是测试代码的基本需求，需要在设计和实现的全过程予以重视。
- 单元测试和系统测试的测试代码具有显著差异。测试人员需要根据所开发的代码，选用恰当的设计和技术。
- 计算机辅助测试的目标是人尽其才（自由发挥测试人员的智能）和物尽其用（充分利用计算机的性能）。
- 掌握多种开发技术能够帮助测试人员高效地完成多样化的测试任务。
- 利用大规模自动化测试，测试人员可以开发出崭新的测试策略，完成其他技术难以达成的测试目标。

第7章

研究产品

James Bach是语境驱动测试学派[1]的奠基人之一。他在该领域进行了持续地研究和实践，并通过课程“快速软件测试”[2]将研究成果传播给测试社区。在他的知识体系中，启发式测试策略模型HTSM（参见4.2.1节）是基础性的概念框架，用来组织各种测试相关的因素和技术。该模型将项目语境抽象为4个因素：产品元素、项目环境、质量标准、测试技术，并通过产品元素、项目环境和质量标准来驱动测试技术。

HTSM提醒测试人员，软件测试需要综合考虑产品元素、项目环境、质量标准和测试技术。所谓“优秀的测试”并不是使用最前沿、最高级的测试技术，而是根据产品和项目的实际情况选择恰当的测试方法。为了制定出符合项目语境的测试策略，本章将讨论如何从测试视角来研究产品，下一章将讨论如何研究项目环境。本章的主要内容包括软件实现分析、软件业务研究和背景领域研究。

7.1 静态分析

静态分析是在不运行软件的情况下，通过阅读源代码来研究产品的活动。对于测试人员而言，阅读代码将帮助他更好地理解产品实现和代码变更，以产生更好的测试想法。本节讨论静态分析在测试活动中的常见应用，并介绍一些基本实践方法。”

7.1.1 浏览源代码来理解产品实现

刚接触一个项目时，测试人员可以浏览产品源代码来掌握其基本情况。由于许多项目的源代码在十万行以上，有些项目甚至达到数百万行，所以测试人员仔细阅读所有代码是不现实的。合理的策略是浏览源代码树，了解源代码的目录结构、模块划分、代码分布等基本情况，然后阅读部分源代码来领会编码规范、架构设计、代码实现等技术细节。在今后的工作中，测试人员可以根据具体的测试任务再去阅读相关的代码。

① http://context-driven-testing.com/。

② http://www.satisfice.com/info_rst.shtml。

假设，测试人员小张加入单元测试框架NUnit[①]的项目小组。他打算花半天的时间来阅读源代码，以概览产品的情况。首先，他浏览NUnit的源代码树来熟悉代码组织。基本方法如下。

- **阅读前要确定阅读目标，但阅读时不要拘泥于预定的目标**。漫无目的地浏览代码会浪费大量的时间，而围绕具体目标来读代码能让阅读者更有效地利用时间，并获得预期的成果。不过，阅读者也可以暂时偏离预定的方向，研究某些他临时产生的想法，或研读一些他觉得很有趣的代码。他可以设定一个15分钟的时间盒去研究这些临时目标，当时间耗尽，他会记录研究成果和值得深入探索的问题，然后回到原定目标上继续阅读。这样进入一个主题再退出的策略被称为浸入与退出，其优点是经过几轮的浸入与退出，阅读者可以渐渐明了产品的结构和模式，并形成更系统的研究策略[Kaner01]。
- **通过目录识别模块**。通常，一个目录对应一个模块，目录的父子关系表示了模块间的隶属关系。阅读者可以用目录名推测模块的大致功能，然后阅读目录中的几个源文件来了解更多的信息。
- **通过模块了解产品的组成结构，知晓各部分代码的功能**。熟悉这些信息有助于评估一个代码变更集所涉及的模块、所影响的功能。
- **借助搜索引擎来了解相关技术**。如果在阅读过程中遇到一些不熟悉的技术，阅读者可以用搜索引擎来获得它们的资料，并通过快速阅读来大致了解它们的背景。
- **一边阅读，一边记录笔记**。阅读代码时应该记录所发现的重要信息，否则大脑很可能在几天后就将它们遗忘。笔记的形式可以多种多样，清单、表格、流程图、信手涂鸦皆可，只要能帮助阅读者理解代码即可。因为源代码已经承载了所有细节，所以笔记内容不必追求完备，只需记录阅读者抽象出的要点即可。

经过一段时间的阅读，小张获得了如下笔记。这些笔记记录了NUnit源代码树的结构（如图7-1所示）和各个目录的主要作用。

- doc：HTML格式的文档（主页为index.html），内容面向NUnit的使用者。
- install：一组Windows Installer XML（WiX）格式的XML文件，定义了所生成的Windows安装文件（MSI）需要包含的程序集。
- lib：NUnit依赖的程序集，包括log4net.dll、Rhino.Mocks.dll和NSubstitute.dll。
- nuget：一组NuGet格式的XML文件，帮助Visual Studio插件NuGet将NUnit加入Visual Studio项目。
- samples：一组使用NUnit的样例代码，所使用的语言包括C++/CLI、C#、F#、VB.NET。
- scripts：一组支持NAnt的XML文件。
- src：NUnit的源代码和测试代码。
- ClientUtilities：一批NUnit的辅助类。

① http://nunit.org/。

- ConsoleRunner：运行用NUnit编写的单元测试的控制台程序。
- GuiComponents：基于Windows Form技术的图形界面组件，包括C#源代码和资源文件。
- GuiException：一批图形界面的辅助类，用于显示单元测试所发现的错误。
- GuiRunner：运行用NUnit编写的单元测试的图形界面程序。
- NUnitCore： NUnit的核心接口和类，包括TestFixture、TestMethod、Logger等。
- NUnitFramework： NUnit的框架代码，包括属性、约束、异常等。
- NUnitMocks：支持虚拟对象的接口和类。
- NUnitTestServer： NUnit的测试服务器（nunit-agent.exe）的代码。
- PNUnit： NUnit扩展PNUnit（Parallel NUnit）的代码。
- ProjectEdit： NUnit项目编辑器（nunit-editor.exe）的代码。
- tests：用NUnit编写的NUnit的测试用例。
- tools：支持NUnit开发的一组工具，包括编译工具NAnt、虚拟对象程序库NSubstitue、安装文件生成工具WiX。

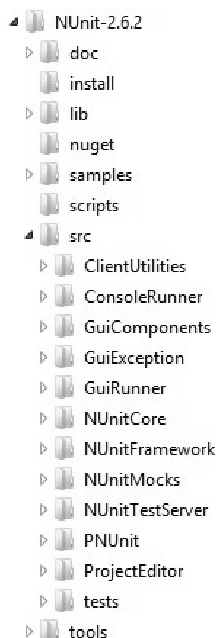


图7-1 NUnit的源代码树和代码组织

由于小张希望尽快了解NUnit的核心代码，所以他标记了以下目录，作为深入阅读的目标。

- samples：阅读测试样例来理解如何使用NUnit。
- ConsoleRunner：阅读控制台程序的代码来理解如何运行NUnit的测试用例。控制台程序比图形界面程序简单，有助于更快地理解NUnit的运作机制。

- NUnitCore：阅读NUnit的核心代码来理解基本概念和实现。
- NUnitFramework：阅读框架代码来理解NUnit如何组织测试用例。

在一览全局并标记重点之后，小张用代码清单7-1所示的PowerShell脚本来分析源代码树。该脚本分析当前目录和子目录下所有C#文件，并输出文件名、文件所在目录和文件行数到控制台。小张将输出结果重定向到文本文件，再用Microsoft Excel打开该文本文件，并生成如图7-2所示的数据表。该表提供了所有源代码文件的行数，为进一步的分析提供了基础。

代码清单7-1　统计文件行数的PowerShell脚本

```
$root = Get-Location
"Folder`tFile`tLineCount"
Get-ChildItem *.cs -Recurse | %{
    $folderName = $_.DirectoryName.Substring($root.Path.Length);
    $lineCount = (Get-Content $_.FullName).Length
    "$folderName`t${$_.Name}`t$lineCount"
}
```

	A	B	C
1	Folder	File	LineCount
2	\ClientUtilities\tests\resources	TestResource.cs	17
3	\ClientUtilities\tests	AssemblyListTests.cs	119
4	\ClientUtilities\tests	AssemblyWatcherTests.cs	133
5	\ClientUtilities\tests	CategoryManagerTest.cs	80
6	\ClientUtilities\tests	CategoryParseTests.cs	144
7	\ClientUtilities\tests	DomainManagerTests.cs	96
8	\ClientUtilities\tests	EventDispatcherTests.cs	239
9	\ClientUtilities\tests	FileWatcherTests.cs	130
10	\ClientUtilities\tests	MemorySettingsStorageTests.cs	104

图7-2　NUnit的源代码行数

代码行数是一种最简单的代码度量，不能提供丰富的语义信息，但是不失为很好的分析切入点。第一，一段很简单的脚本就可以统计出源代码的行数，让阅读者能够立即了解代码的情况。第二，任何度量方法都只反映特定方面的信息，代码行数简单明了，较一些复杂的方法更直接地说明了代码的情况。第三，理解产品实现需要仔细研读代码，任何度量方法都是辅助。既然如此，不妨从最简单的度量入手。

基于如图7-2所示的数据表，小张获得了代码行数的统计结果（详见表7-1），并绘制出代码行数的分布图（详见图7-3）。由表7-1可知，NUnit的测试代码要远多于产品代码，测试代码的文件数是产品代码的4.2倍，测试代码的代码行数是产品代码的4.4倍，这暗示NUnit很可能采用测试驱动开发，自动化测试的代码量很大。小张还看出NUnit的产品代码有2万多行，平均到每个文件只有145行。这说明NUnit是一个小型的测试框架，大多数源代码文件长度较短。图7-3也确认大部分源代码文件的长度小于500行，超过1000行的文件屈指可数。

表7-1 NUnit的代码行数统计

	文件数	代码行数	代码行数/文件数
产品代码	158	22944	145.2
测试代码（文件名以Tests.cs结尾）	674	102534	152.1
总和	832	125478	150.8

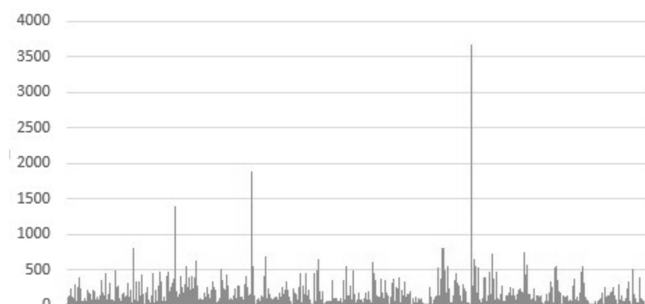


图7-3 源代码文件（产品代码和测试代码）的代码行数分布

随后，小张绘制出图7-4，以获悉代码行数最多的10个文件。由该图可知，代码最多的是Assert.cs。通过阅读代码（详见图7-5），小张发现Assert.cs包含一批测试断言函数。断言函数数量庞大且函数注释较长，该文件达到3600多行。但是每个函数都很简单，所以整个文件的复杂度并不高。接下来3个代码较多的文件与图形界面相关，说明图形界面是NUnit中相对复杂的部分。然后是测试加载器TestLoader.cs，阅读其代码可知它帮助测试运行器加载和卸载测试用例。最后是3个约束相关的文件，这暗示约束是NUnit中一个较复杂的概念。

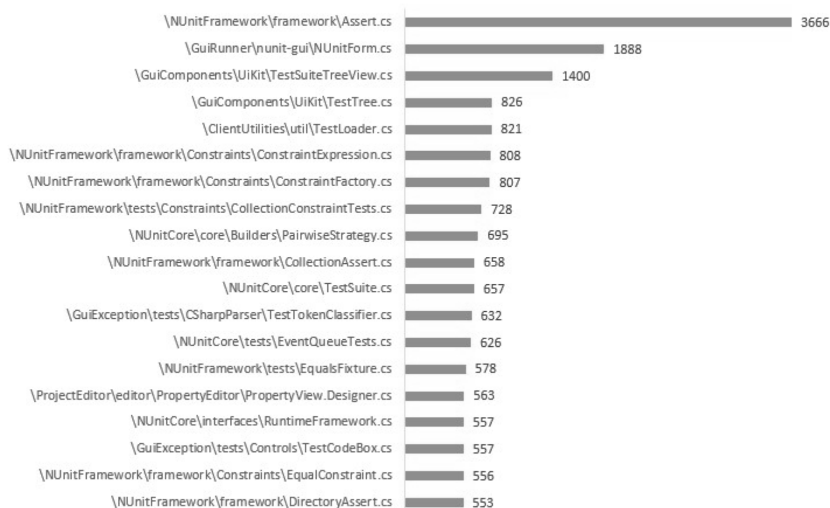
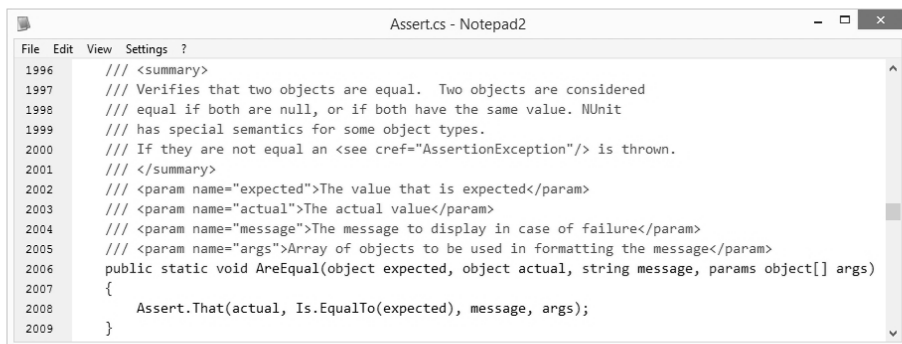


图7-4 代码行数最多的10个文件



Assert.cs - Notepad2

File　Edit　View　Settings　?

```
1996    /// <summary>
1997    /// Verifies that two objects are equal.  Two objects are considered
1998    /// equal if both are null, or if both have the same value. NUnit
1999    /// has special semantics for some object types.
2000    /// If they are not equal an <see cref="AssertionException"/> is thrown.
2001    /// </summary>
2002    /// <param name="expected">The value that is expected</param>
2003    /// <param name="actual">The actual value</param>
2004    /// <param name="message">The message to display in case of failure</param>
2005    /// <param name="args">Array of objects to be used in formatting the message</param>
2006    public static void AreEqual(object expected, object actual, string message, params object[] args)
2007    {
2008        Assert.That(actual, Is.EqualTo(expected), message, args);
2009    }
```

图7-5　Assert.cs源代码（局部）

为了更好地理解模块的代码量，小张又绘制了图7-6，以了解代码行数最多的10个目录。由该图可知，图形界面模块UiKit拥有最多的代码，其次是Constraints、core、framework、util、interface等NUnit的核心模块。随后是约束的测试代码（\NUnitFramework\tests\Constraints），这再次暗示约束可能是比较复杂的概念，拥有较多的产品代码和测试代码。

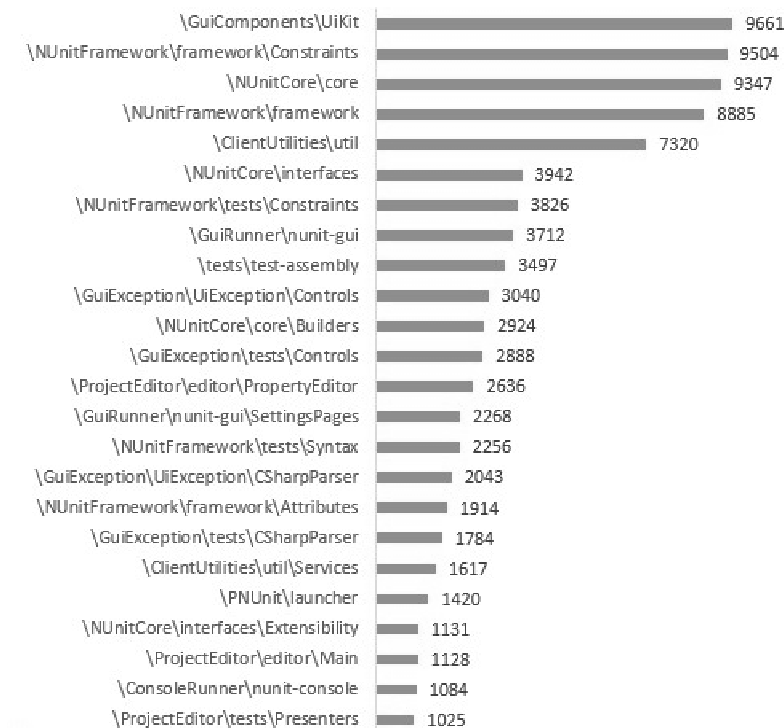


图7-6　代码行数最多的10个目录

根据表7-1、图7-3、图7-4和图7-6，小张获得了NUnit的代码分布，了解了模块和文件的代码

量，为深入研究提供了基本信息。如果熟悉脚本语言和电子表格，获得统计信息并作图只需十多分钟的时间，而且还可以绘制更多的图表，从不同的角度挖掘源代码的信息。

回顾以上代码度量步骤不难获得如下经验。

- 代码度量提供了代码某个方面的概况，可以帮助阅读者一览全局。
- 简单的代码度量很容易实现。本节的案例表明不使用语法分析、仅利用文本分析就能提供一些有帮助的信息。
- 用恰当的工具（电子表格、数据库等）分析数据，可以快速地计算并展示数据。这提高了数据分析的效率。
- 代码度量只提供了代码阅读的线索，获得更透彻地理解仍旧需要深入阅读源代码。

在知晓代码的基本情况后，小张开始阅读具体的代码，来获得更深刻的理解。通常，阅读者没有时间查看所有源代码，因此他需要选择性地阅读。基本的策略是选择与当前测试任务有关的代码。因为小张的阅读目标是熟悉NUnit代码，所以他打算阅读一些核心的且自己感兴趣的代码。他所选择的代码是约束。

小张将约束的测试用例作为阅读的切入点，利用测试代码来理解约束的用法和语义。考虑到NUnit提供了很好的文档，小张决定从文档入手。他通过文档的首页（doc\index.html）发现了一批约束相关的页面（详见图7-7）。他快速浏览这批页面，重点阅读文档摘要和代码示例，从而逐渐形成以下想法。

- 约束的全称应该是“基于约束的断言模型”，它由一组类构成，用于编写测试检查代码。
- 利用约束编写的断言代码类似于测试领域专属语言，让测试开发人员可以用精炼的代码来表达丰富的语义。例如，语句`Assert.That（intArray, Has.All.GreaterThan(0)）`检查整数数组`intArray`的所有元素都大于0。该语句的核心是约束代码`Has.All.GreaterThan(0)`，它用短小清晰的格式表达出检查方法，让测试逻辑一目了然。
- 基于约束的断言代码包括两部分：约束的使用者（如断言函数`Assert.That(object actual, IResolveConstraint constraint)`）和约束的实现者（实现接口IResolveConstraint的对象）。
- 从`Assert.That`的函数签名推测，约束的实现手法类似于命令模式 [Gof94]，即测试代码定义命令，然后将命令传递给`Assert.That`，由后者调用命令。
- 约束的呈现方式是一个表达式（如`Has.All.GreaterThan(0)`）。其实现方式可能是解释器模式 [Gof94]，即通过表达式树来定义计算逻辑。

为了检验对代码实现的推测，小张搜索源代码，发现了`Assert.That(…)`对`IResolveConstraint`的调用（详见图7-8）和`IResolveConstraint`的定义（详见图7-9）。由这两段代码，他基本确认约束使用者与约束实现者通过命令模式来连接。然后，他浏览约束实现类的源代码（位于目\src\NUnitFramework\framework\Constraints）。根据`ConstraintExpression`（详见图7-10）等类的实现代码，他基本确定约束的逻辑由表达式树来实现。

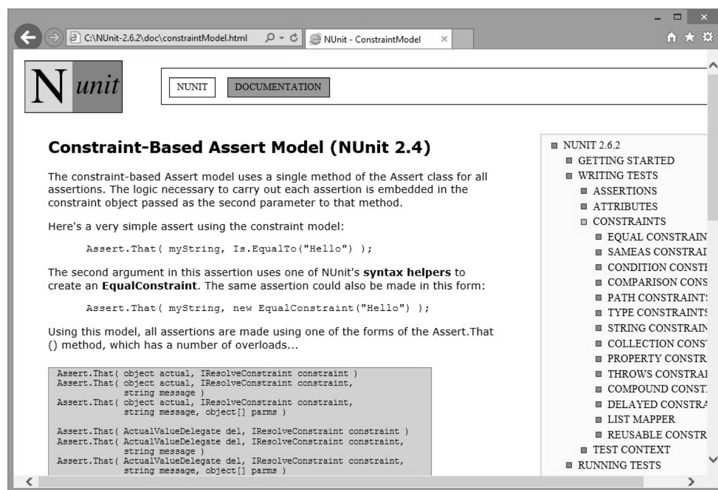


图7-7　约束的文档

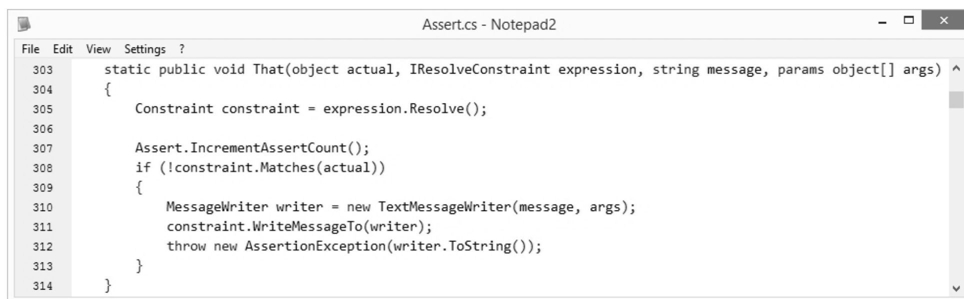


图7-8　Assert.That对IResolveConstraint的调用

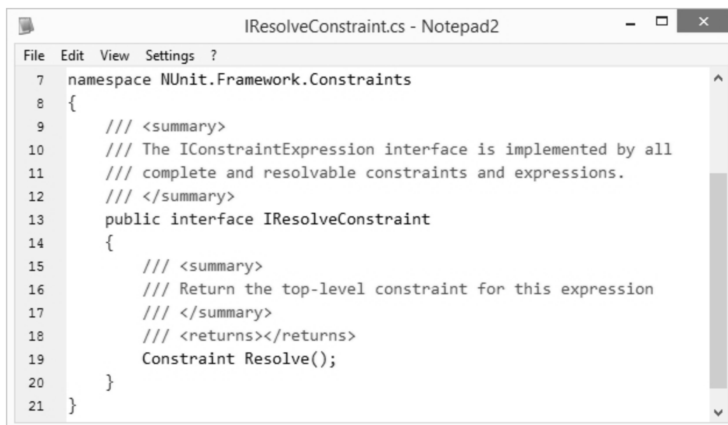


图7-9　IResolveConstraint的定义

```
ConstraintExpression.cs - Notepad2
File  Edit  View  Settings  ?
10  namespace NUnit.Framework.Constraints
11  {
12      /// <summary>
13      /// ConstraintExpression represents a compound constraint in the
14      /// process of being constructed from a series of syntactic elements.
15      ///
16      /// Individual elements are appended to the expression as they are
17      /// reognized. Once an actual Constraint is appended, the expression
18      /// returns a resolvable Constraint.
19      /// </summary>
20      public class ConstraintExpression : ConstraintExpressionBase
21      {
22          /// <summary>
23          /// Initializes a new instance of the <see cref="T:ConstraintExpression"/> class.
24          /// </summary>
25          public ConstraintExpression() { }
26
```

图7-10　ConstraintExpression的定义

由以上代码阅读过程，不难提取出一些代码阅读的基本实践。

- 阅读者针对一个特定的主题进行阅读，以点带面地推动代码理解。
- 对于一组代码，阅读者需要从外部代码的角度理解它们的功能和用法，并评估它们对整个产品的贡献。
- 对于一组代码，阅读者需要理解它们的实现方法，包括所使用的设计、所采用的技术、所依赖的其他代码等。
- 了解语言特性、惯用设计、命名规范等基础知识会帮助阅读。阅读者需要主动地学习和积累这方面的知识。
- 代码阅读并没有固定的阅读顺序。阅读者可以由一个函数或类切入，搜索并阅读它的调用代码和实现代码。他也可以顺序浏览一个目录中的源代码文件，以通盘了解。
- 在阅读过程中，阅读者会产生一些想法。应该将它们记录下来，并阅读更多的代码来检验这些想法，从而获得更深刻的理解。

7.1.2　分析源代码来帮助测试设计

在具体的测试任务中，测试对象通常是一组代码变更。它们或增加产品的功能，或重构已有的代码，或修复已知的缺陷。其特征都是只改动产品的部分代码，并要求修改后的产品可以正常工作。为了更有效的测试，测试人员需要阅读代码来以理解变更的影响，并激发测试想法。

影响分析是一种常用的代码分析方法，旨在分析目标代码受哪些代码的影响，又会影响到哪些代码。对于测试任务而言，测试人员主要关注以下两个方面。

- 哪些代码会影响被修改代码？被修改代码的输入数据从何而来？如何构造测试用例让程序执行覆盖被修改代码？如何构造测试用例以测试被修改代码的状态？能否构造测试用例让被修改代码发生错误？

- 被修改代码会影响到哪些代码？被修改代码的输出数据流向何处？如何观察到被修改代码的输出？如何构造测试用例以更全面地测试受影响的代码？能否让被修改代码产生"异常"的输出数据，令受影响的代码发生错误？

为了识别代码之间的影响，测试人员需要在阅读代码的过程中实施控制流分析和数据流分析。

- **控制流分析检查语句A的执行是否影响语句B的执行**。以下是一些基本的分析规则。
  - 如果函数A调用函数B，那么函数A控制影响函数B，也称函数B控制依赖于函数A。
  - 分支语句的条件表达式控制分支语句的子语句。例如，在`if (expr) { A; } else { B; }`中，表达式`expr`控制影响语句A和语句B，也称语句A和语句B控制依赖于表达式`expr`。
  - 循环语句的条件表达式控制循环语句的循环体。例如，在`while (expr) { A; }`中，表达式`expr`控制影响语句A，也称语句A控制依赖于表达式`expr`。
  - 跳转语句所跳过的语句依赖于跳转语句。例如，在`if (expr) { return; } A;`中，跳转语句`return`改变了程序的执行流，执行`return`语句将导致语句A不被执行。于是，`return`语境控制影响语句A，也称语句A控制依赖于`return`语句。

- **数据流分析检查语句A赋值的数据是否被语句B使用**。
  - 如果变量x被语句A赋值或修改，那么称语句A是变量x的定义点。
  - 如果变量x的值被语句B使用，那么称语句B是变量x的引用点。
  - 如果语句A是变量x的定义点，语句B是变量x的引用点，存在一条从A到B的执行路径，且该路径上x不被修改，那么称语句A数据影响语句B，也称语句B数据依赖于语句A。

接下来通过一个实例来展示影响分析在代码阅读中的运用。程序wc[①]是Unix平台的一个常用命令，用来统计文本文件的字符数、单词数和行数。图7-11展示了wc最基本的函数调用关系[②]。主函数`main`从命令行接受待分析的文件名，然后调用函数`counter`完成统计任务。为了统计字符、单词和行的数目，`counter`调用函数`getword`，而后者调用函数`isword`来判断一个字符是否为单词字符。



图7-11　wc.c中的函数调用关系

假设函数`isword`的实现代码被修改，测试人员需要检查该代码变更没有引入错误。他在源代码文件wc.c中搜索`isword`的调用点，发现了它的两个调用点都在函数`getword`中（具体实现请参考代码清单7-2）。然后，他对每一个调用点进行了分析。

- `getword`的第一个调用点在语句C。语句C使用变量c（即语句C是变量c的引用点），该变量

① wc的样例代码：http://www.gnu.org/software/cflow/manual/html_node/Source-of-wc-command.html。
② 完整的函数调用图请参考：http://www.gnu.org/software/cflow/manual/html_node/ASCII-Tree.html#ascii%20tree。

的取值来自于语句B（即语句B是变量c的定义点），故语句C数据依赖于语句B。在语句B中，变量c的值读取自文件指针fp，而文件指针的定义点是语句A。所以，完整的数据流是A→B→C，即被修改代码isword的输入数据来自于fp对应的文件（数据流图请参考图7-12）。这提示测试人员构造各种各样的文件以周密地测试代码变更。

- 构造一个或一组文件覆盖isword的输入参数c的所有取值，即unsigned char的所有取值（256个值）。
- 为了检查wc的正确性，构造一个或一组文件包含空格、Tab键、数字、字母、回车、换行等字符，以检查wc正确地统计了字符数、单词数和行数。
- 测试ASCII和Unicode编码的文件。
- 测试文本文件和二进制文件。

- ❑ 在语句C中，getword位于if语句（分支语句）的条件表达式中，它的返回值会影响程序的执行。进一步分析可知，语句D和语句E控制依赖于语句C。进一步地，语句E是跳转语句，如果它被执行，那么语句F不会被执行，故语句F控制依赖于语句E。所以，完整的控制流是C→{D;E}→F→COUNT(c)（控制流图请参考图7-12）。分析控制流上的语句可知，它们会递增变量wcount、ccount和lccount的值。这提示了一些测试想法。

  - 构造测试输入，让wcount、ccount或lccount的值不被修改。
  - 构造测试输入，让wcount、ccount或lccount发生整数溢出。

- ❑ getword的第二个调用点在语句H。经过分析可得数据流A→F→H和控制流H→E→F（详情请参考图7-12）。之后，测试人员可实施测试分析，以获得测试想法。

代码清单7-2　wc.c中getword和isword的代码

```
/* Return true if C is a valid word constituent */
static int
isword (unsigned char c)
{
    return isalpha2 (c);            // 此处代码被修改
}


/* Increase character and, if necessary, line counters */
#define COUNT(c)       \
    ccount++;        \
    if ((c) == '\n') \
        lcount++;



/* Get next word from the input stream. Return 0 on end
    of file or error condition. Return 1 otherwise. */
int
getword (FILE *fp)                // A: 变量fp的定义点
```

```
{
    int c;
    int word = 0;

    if (feof (fp))
        return 0;

   while ((c = getc (fp)) != EOF)  // B: 变量c的定义点，也是变量fp的引用点
   {
       if (isword (c))             // C: 函数isword的调用点，也是变量c的引用点
       {
           wcount++;               // D: 该语句控制依赖于语句C
           break;                  // E: 该语句控制依赖于语句C
       }
       COUNT (c);                  // F: 该语句控制依赖域语句E
   }

   for (; c != EOF; c = getc (fp)) // G: 变量c的定义点，也是变量fp的引用点
   {
       COUNT (c);                  // H: 该语句控制依赖于语句J
       if (!isword (c))            // I: 函数isword的调用点，也是变量c的引用点
           break;                  // J: 该语句控制依赖于语句I
   }

   return c != EOF;
}
```

图7-12展现了以上影响分析的结果，其中实线表示数据流，虚线表示控制流。对于简单的程序，测试人员只要在头脑中分析语句的影响关系即可，无需绘制详细的图形。对于复杂的程序，测试人员可以一边阅读，一边写下重要的语句或函数，并绘制出它们之间的关系。这能够帮助他同时追踪多行代码，发掘它们的关系，从而较完整地理解代码。

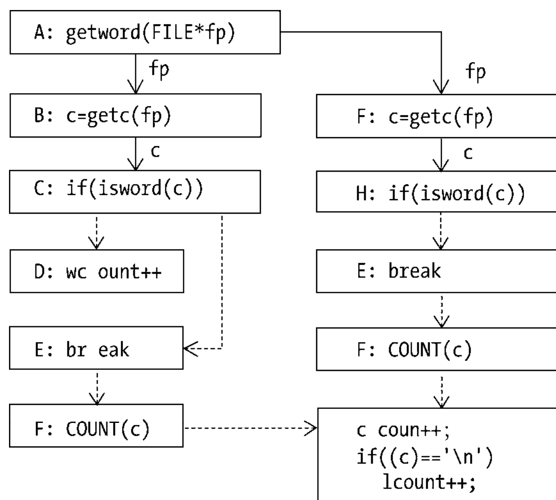


图7-12　函数getword的数据流和控制流（局部）

值得一提的是，全局变量是需要特别关注的对象。它们往往被软件的多个功能读写，会广泛地影响软件的控制流和数据流。因为它们的读写代码通常分布在多个文件中，只阅读变更代码不能完整地理解它们的影响。此时，测试人员需要在整个源代码树上搜索它们的引用点和和定义点，然后分析变更代码如何影响这些引用点、又如何被这些定义点所影响。因为代码修改者很可能没有考虑这些引用点和定义点，所以变更代码会引入某些缺陷，而全面的影响分析将找出这些问题。

与影响分析相似，**污染传播分析**也是一种基于控制流和数据流的代码分析方法[Chess07]。它重点关注数据流，通过分析数据从何处进入软件和数据如何在软件中传播，来发掘隐藏的缺陷。污染传播分析是一些静态代码检查工具的基础，在软件安全领域得到成功的应用，能够发现许多动态测试难以暴露的安全性问题。对于普通的测试人员，即便没有代码检查工具的辅助，他也可以利用污染传播的思想来指导代码阅读，以更好地设计测试。

污染传播的核心隐喻是“污染”和“传播”。所谓“污染”是指被污染的数据，其源头通常是包含恶意数据的攻击性输入。所谓“传播”是指软件没能过滤掉被污染的数据，让这些恶意数据在软件中传播，最终导致软件故障，或将恶意数据传递给用户和其他软件。为了发现此类问题，测试人员可以考虑以下代码阅读方法。

- **标记软件的入口点，即寻找接收外部输入的代码**。这批代码会调用语言、程序库或平台的API（这些API的名字常包含read、receive、accept、fetch等单词）来接受数据，其数据来源包括用户输入、文件、数据库、平台调用、网络通信等。
- **分析以入口点为源头的数据流，寻找输入检查代码**。通常，检查代码的使命是发现异常输入、报告错误并过滤掉异常数据，使污染不能在软件中传播。
  - 如果不存在检查代码，那么测试人员分析使用输入数据的代码，然后构造出攻击性的输入数据，让使用该数据的代码出错。
  - 如果存在检查代码，那么测试人员仔细分析检查代码，以挖掘检查逻辑的漏洞。无论是否发现了确凿的漏洞，他都可以针对检查代码输入攻击性的数据，以测试它能否正确地过滤掉恶意数据。
- **标记软件的出口点，即找到发送输出数据的代码**。这批代码会调用语言、程序库、平台的API（这些API的名字常包含write、send、output、flush等单词）来写出数据，其输出目标包括用户界面、文件、数据库、其他程序、网路服务等。
- **分析以出口点为终点的数据流，寻找输出之前清理数据和格式化数据的代码**。
  - 清理数据的代码应该过滤掉异常数据，使得污染不会传递给其他软件或数据存储（文件、数据库、注册表等）。测试人员的任务是分析这些代码，然后通过测试用例来展开攻击，以检查它能否正确地发现并处理不正确的数据。阅读代码可以提供更多的攻击思路，较单纯的黑盒测试更加高效。

- 格式化数据的代码会转化数据的表现形式，以产生目标软件或数据存储可以接受的数据。不正确的格式化会产生错误的数据，对数据接收者而言这些错误的数据也是一种污染。测试人员的任务是分析格式化代码，然后构造测试用例来暴露它们的问题。
- 尝试寻找一条从入口点到出口点的数据流。在该数据流上，输入检查代码和输出检查代码都存在问题，使得恶意的数据可以从入口点一直传播到出口点，从而影响到其他软件或数据存储。

图7-13摘录自4.2.3节，描述了一个客户端–服务器架构的报表系统。在客户端，一个运行在IE中的Silverlight应用从服务端获得数据，并绘制报表。在服务端，一个同步工具将外部数据源的数据写入数据库，一个Windows服务读取该数据库，并将数据传递给一个IIS网站，最后该网站将数据传递给客户端的Silverlight应用。

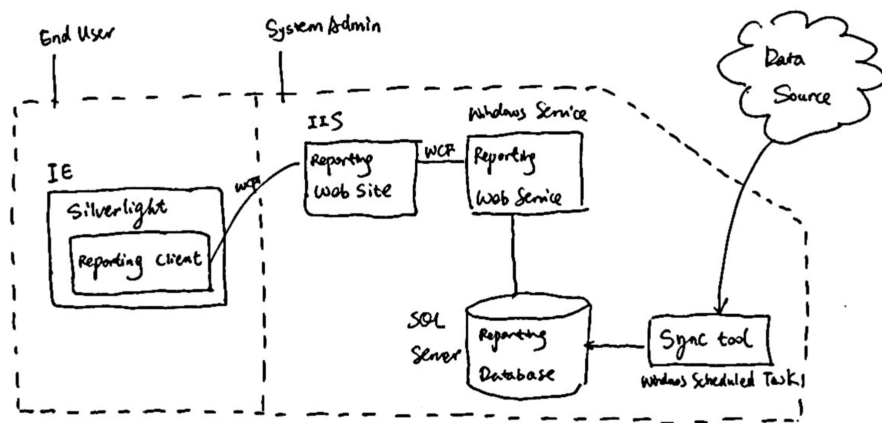


图7-13 报表系统的系统生态图

根据污染传播的思路，测试人员可以考虑如下代码阅读策略。

❑ **阅读同步工具和数据库的代码，并构造测试用例。**

- 检查它能否处理来自外部数据源的异常数据。
- 检查它不会将错误的数据写入数据库。一个典型且严重的错误是同步工具存在SQL注入的安全漏洞，让来自外部数据源的恶意数据篡改或删除了数据库的数据。测试人员不应该假定外部数据源是安全的，因为它可能被其他软件攻击，以致于包含恶意数据。许多安全事故表明，黑客会构造出一条攻击链，将多个软件作为攻击跳板，以攻陷目标软件。
- 数据库的字段往往有固定的长度，例如SQL Server中INT是32位，BIGINT是64位，`NVARCHAR(20)`最多只能容纳20个Unicode字符等。测试人员需要检查同步工具写入的数据是否会超过字段的长度，是否会导致存储过程的计算发生错误。

❑ 阅读Windows服务和IIS网站的代码，并构造测试用例。

- 如果它们所读取的数据库包含错误数据，它们的输入检查代码能否发现错误？它们能否将错误信息用合理的方式报告给最终用户和系统管理员？
- 数据库提供了符合格式和范围的数据，它们的报表计算代码是否正确？在计算过程中会不会产生数值溢出、报表超出范围、排版错误等问题？报表计算代码能否发现这些错误，并用合理的方式报给最终用户和系统管理员？
- 它们的输出检查代码能否正确工作？错误的数据和报表会不会传递给客户端的Silverlight应用？这些被污染的数据会不会破坏客户端应用，导致用户数据丢失？
- 它们的输入检查代码能否正确工作？如果客户端提交了恶意数据，数据库会不会被篡改？测试人员不应该假设客户端程序是安全的，因为黑客可以编写一个程序，伪装成客户端应用向IIS网站提交HTTP请求。

❑ 阅读客户端程序代码，并构造测试用例。

- 客户端的输入包括用户提交的数据和IIS网站给出的报表。
- 测试人员检查用户提交的错误数据不会被传递给IIS网站，即不存在从用户到网站的污染传播路径。这通常要求客户端程序对用户的输入做充分的检查。
- 测试人员检查IIS网站给出的报表不会导致客户端程序崩溃或其他严重问题。这通常要求客户端程序有足够好的异常处理功能。有时，网站会给出错误的报表或非常复杂的报表，让客户端程序不能完成数据展示任务。此时，客户端程序应该妥当地结束计算，并向最终用户报告错误。

在阅读代码的过程中，测试人员会发现一些缺陷。更多的时候，他会发现一些“感觉有问题”的代码，但并不能确定它们一定存在缺陷。这时，他可以咨询程序员，参考他们的解释和推测，但应该保持合理的怀疑。然后，他会构造测试用例来攻击这些疑似有问题的代码，让测试结果来回答心中的疑问。对于测试人员而言，阅读代码的主要目的不是代码审查（阅读代码所发现的缺陷是副产品），而是发现所有的测试对象，以避免测试遗漏。在此基础上，分析测试对象，构造有针对性的测试想法，提高测试执行的效率，才是代码阅读的基本出发点。

7.1.3 黑盒测试并不是基于无知的测试

测试人员是否应该阅读产品的源代码在测试社区有争议。有些测试人员认为阅读代码不利于有效的测试，其主要见解有两点。

第一，阅读源代码可能会误导测试人员。当测试人员阅读代码时，他有可能产生“这段代码看上去没问题”或“这段代码变更只会影响模块A”等看法。因为软件是高度复杂的，单纯地阅读代码不能准确预测其运行时的行为，所以测试人员的看法可能是错误的。如果他根据错误的想

法来实施测试或认为没有必要测试，那么他会遗漏一些原本可以发现的缺陷。

第二，阅读源代码可能会束缚测试人员的思路。阅读源代码让测试人员的思维方式更接近程序员，从而丧失了黑盒测试的重要优势：测试人员与程序员的思考角度不同，因此有可能发现程序员所遗漏的缺陷[Kaner01]。此外，黑盒测试要求测试人员理解软件的用户，用他们的视角考察软件。这有助于交付真正有价值的软件，确立产品的竞争优势。局限于代码实现的测试思考，将错过一些重要的测试活动和产品信息，不利于提高产品质量。

我认为以上两点都准确描述了测试人员只依赖源代码进行测试的风险。即便如此，阅读源代码仍旧是一项重要且必要的测试活动，主要原因有以下四点。

第一，因为软件是高度复杂的，任何方法都不能获得完整的产品信息，所以测试人员应该多角度地研究软件，运用多种手段去调查产品的质量。恰如科学家不会刻意排除某种研究方法、侦探不会故意放过某条线索，测试人员也不该先入为主地放弃任何一种调查方法。

第二，源代码是软件产品最重要的文档，提供了许多无可替代的产品信息。研究源代码和软件架构能够帮助测试人员更全面地理解产品，从而设计出有针对性的测试。7.1.1节的案例说明，阅读源代码有助于测试人员理解产品，并独自调查一些感兴趣的主题。7.1.2节的案例表明，分析源代码可以产生许多测试想法，这些测试想法针对产品的实现，有很强的查错能力。

第三，研究产品实现有助于避免测试设计的缺陷。在紧张的项目过程中，测试人员可能会犯一些错误，例如测试活动没有覆盖需要测试的对象、测试方法不适合产品的特征、测试用例没有检查重要的实现细节等。因此，测试人员需要一些方法来评估并改善测试设计。阅读源代码帮助测试人员建立产品的实现模型，了解对测试设计有重要影响的代码细节，可以有效地提高测试设计的质量。例如，在一轮黑盒测试后，测试人员搜索代码，列出所有尚未测试的入口点。这些入口点说明先前的测试设计存在漏洞，他需要补充设计一批测试用例来覆盖它们。

第四，降低代码阅读的风险并不困难。代码阅读的典型风险是误导测试设计和约束测试人员思维。好的测试人员会培养出从多个角度考察产品、从多个信息源获取信息的习惯，自然而然就降低了此类风险。此外，一个有帮助的启发式规则是：阅读代码所获得的“想法”只是一种“假说”，需要通过实验（设计并运行测试用例）来验证。在阅读代码时，测试人员会对代码产生一些判断。他需要提醒自己，这些观点尚未得到验证，应该设计测试来检查其正确性。这样就将代码阅读和测试设计联系在一起，从而让它们相互支持、彼此促进。

测试专家Cem Kaner等指出“黑盒测试并不是基于无知的测试”。测试人员对产品理解得越好，调查产品的方式越多，就越有可能更好地测试 [Kaner01]。阅读源代码是一项重要的技术调查方法，可以为测试设计提供想法和反馈，与其他测试技术一起运用，能够帮助测试人员更有效地完成测试任务。

7.2　动态分析

动态分析是通过运行软件来研究产品实现的活动。虽然动态分析只覆盖了一部分代码执行路径，它能详细检查产品在这些路径上的状态，对于理解特定情景和设计有事半功倍之效。本节将介绍一些动态分析技术在测试中的应用。

7.2.1　用工具分析产品的行为

深入的测试要求测试人员透彻地了解软件的动态行为。一种常见的策略是先建立产品的概念架构，从宏观上理解其行为，然后根据测试任务深入研究它的细节，并产生一些测试想法。在此过程中，合理地使用调试和诊断工具可以提高技术调查的效率。本节将讨论一组测试人员经常执行的动态分析任务，并介绍相关的分析方法和技巧。

任务1：列出产品所加载的动态链接库

为了调用程序库、框架、操作系统的功能，软件的可执行程序会加载相应的动态链接库。查看产品所加载的动态链接库能了解产品的实现技术，为深入的测试提供线索。

例如，测试人员小张为了更好地理解必应词典桌面版[①]（2.0版），使用Process Explorer[②]来列出它加载的动态链接库。他在Process Explorer中选中必应词典的进程BingDict.exe，按下组合键【Ctrl+D】以显示该进程加载的动态链接库。他点击“路径”列的表头，让动态链接库列表按照其路径排序，获得如图7-14所示的结果。通过浏览该列表，小张可以了解产品从哪里加载动态链接库，推测它们的用途，并产生相应的测试想法。

Name	Description	Version	Path
tiptsf.dll	Touch Keyboard and Handwriting Panel Text Ser...	6.2.9200.16433	C:\Program Files (x86)\Common Files\Microsoft Shared\Ink\tiptsf.dll
msado15.dll	ActiveX Data Objects	6.2.9200.16384	C:\Program Files (x86)\Common Files\System\ado\msado15.dll
msadrh15.dll	ActiveX Data Objects Rowset Helper	6.2.9200.16384	C:\Program Files (x86)\Common Files\System\ado\msadrh15.dll
oledb32.dll	OLE DB Core Services	6.2.9200.16384	C:\Program Files (x86)\Common Files\System\Ole DB\oledb32.dll
oledb32r.dll	OLE DB Core Services Resources	6.2.9200.16384	C:\Program Files (x86)\Common Files\System\Ole DB\oledb32r.dll
BingDict.exe	微软必应词典	2.0.0.17017	C:\Program Files (x86)\Microsoft Bing Dictionary\BingDict.exe
DefMgr.dll			C:\Program Files (x86)\Microsoft Bing Dictionary\DefMgr.dll
mfc100u.dll	MFCDLL Shared Library - Retail Version	10.0.40219.325	C:\Program Files (x86)\Microsoft Bing Dictionary\mfc100u.dll
msvcp100.dll	Microsoft® C Runtime Library	10.0.40219.325	C:\Program Files (x86)\Microsoft Bing Dictionary\msvcp100.dll
msvcr100.dll	Microsoft® C Runtime Library	10.0.40219.325	C:\Program Files (x86)\Microsoft Bing Dictionary\msvcr100.dll
NewWordListv1.6Helper.dll			C:\Program Files (x86)\Microsoft Bing Dictionary\NewWordListv1.6Helper.dll
sqlceer35EN.dll	Native Error Strings and Resources	3.5.8080.0	C:\Program Files (x86)\Microsoft SQL Server Compact Edition\v3.5\sqlceer35...
sqlceoledb35.dll	OLEDB Provider (32-bit)	3.5.8080.0	C:\Program Files (x86)\Microsoft SQL Server Compact Edition\v3.5\sqlceoledb..
sqlceqp35.dll	Query Processor (32-bit)	3.5.8080.0	C:\Program Files (x86)\Microsoft SQL Server Compact Edition\v3.5\sqlceqp35...
sqlcese35.dll	Storage Engine (32-bit)	3.5.8080.0	C:\Program Files (x86)\Microsoft SQL Server Compact Edition\v3.5\sqlcese35...

图7-14　按路径排序的必应词典的动态链接库（局部）

- BingDict.exe从Ink目录加载了tiptsf.dll。由动态链接库的描述可知，它与触摸键盘和手写

① http://bing.msn.cn/dict/。

② http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx。

输入有关。这提示小张需要准备一台具备触摸屏的计算机，以测试必应词典能否完善地支持触摸输入。

- ❑ BingDict.exe加载了ADO、OLE DB、SQL Server Compact Edition相关的动态链接库。这暗示必应词典会访问数据库。从软件功能的角度，必应词典的“生词本”需要保存和读取大量数据，最有可能使用数据库。其他功能只需存储少量数据或只需读取数据，都不需要数据库的支持。小张记录他的发现和猜测，以便稍后向程序员询问具体实现细节。
- ❑ 因为必应词典使用数据库，小张自然想到了一些测试想法，并记录如下。在他了解更多实现细节后，他会设计并执行更多的测试。
  - 如果数据库文件丢失或损坏，产品能否妥善地处理？
  - 生词本的词条有没有数量的限制？如果有限制，上限是多少？如果没有限制，大量的词条会不会导致产品性能下降？
  - 生词本在记录词条时，对生词的长度有没有限制？如果生词或词组的长度超过了限制，产品能否妥善地处理？在数据库设计上，还有没有类似的限制？产品又如何处理超出限制的情况？
  - 内存紧张、磁盘耗尽等情况可能导致数据库读写操作抛出异常。产品能否妥善处理这些异常？
- ❑ BingDict.exe从必应词典的安装目录加载了一些动态链接库，其中包括MFC和C语言运行库的动态链接库。这暗示必应词典2.0版用C/C++编写，其用户界面基于MFC。小张记录该发现，以提示自己在今后的测试中关注内存泄漏、缓冲区溢出、悬挂指针等C/C++程序的常见问题。

然后，小张点击“描述”列的表头，让动态链接库列表按照其描述排序，获得图7-15所示的结果。按描述浏览动态链接库，有助于快速发现相关的组件，即便它们位于不同的目录。例如，图7-15显示必应词典加载了IE浏览器的动态链接库（ieframe.dll、ieframe.dll.mui、wininet.dll、jscript.dll、jscript9.dll）。对此，小张记录了一组测试想法。

sechost.dll	Host for SCM/SDDL/LSA Lookup APIs	6.2.9200.16384	C:\Windows\SysWOW64\sechost.dll
ieframe.dll	Internet Browser	10.0.9200.16519	C:\Windows\SysWOW64\ieframe.dll
ieframe.dll.mui	Internet Browser	10.0.9200.16384	C:\Windows\SysWOW64\en-US\ieframe.dll.mui
wininet.dll	Internet Extensions for Win32	10.0.9200.16519	C:\Windows\SysWOW64\wininet.dll
IPHLPAPI.DLL	IP Helper API	6.2.9200.16420	C:\Windows\SysWOW64\IPHLPAPI.DLL
winmm.dll	MCI API DLL	6.2.9200.16384	C:\Windows\SysWOW64\winmm.dll
mfc100u.dll	MFCDLL Shared Library - Retail Version	10.0.40219.325	C:\Program Files (x86)\Microsoft Bing Dictionary\mfc
mshtml.dll	Microsoft (R) HTML Viewer	10.0.9200.16525	C:\Windows\SysWOW64\mshtml.dll
jscript.dll	Microsoft ® JScript	5.8.9200.16519	C:\Windows\SysWOW64\jscript.dll
jscript9.dll	Microsoft ® JScript	10.0.9200.16519	C:\Windows\SysWOW64\jscript9.dll

图7-15　按描述排序的必应词典的动态链接库（局部）

- ❑ 当前计算机的操作系统是Windows 8，必应词典所加载的动态链接库属于IE10。在Windows 7和更早的操作系统上，必应词典能否正确加载IE9、IE8、IE7等浏览器的动态链接库？

- ❑ 在欧洲销售的Windows系统不预装任何浏览器。如果必应词典不能加载IE的动态连接库，它能否正确地向用户报告情况？
- ❑ 在默认情况下，显示网页的IE10进程的完整性级别是低，这有助于提高操作系统的安全性，并保护用户数据。然而，必应词典的进程完整性是中，这会不会带来安全性风险？

在分析动态链接库时，测试人员还需要留意以下情况。

- ❑ **产品所加载的动态链接库可能被其他软件所使用**。当升级这些软件时，动态链接库可能被升级到更新的版本。如果新版本的动态链接库存在向后兼容的缺陷，被测产品可能发生故障。此外，卸载这些软件可能删除动态链接库，从而导致产品不能正常工作。测试人员需要找到并记录这批动态链接库。它们通常位于Windows目录下，其发布者不是微软公司。根据这些动态链接库，测试人员可以定位影响它们的软件，然后拟定一份兼容性测试计划。该测试会检查产品与这些软件的主流版本可以在一台计算机上共存，并且它们的安装和卸载不会破坏彼此的功能。
- ❑ **产品会在执行过程中动态地加载或卸载动态链接库**。测试人员可以用Windbg等工具监视产品的运行，了解它在启动之后又加载了哪些动态链接库。此外，在执行了一个重要的操作之后，测试人员可以用Process Explorer观察产品的动态链接库列表，发现新增的动态链接库。这些链接库通常与之前的操作紧密相关，是测试该操作需要考虑的因素。

任务2：列出产品打开的文件

为了实施文件漫游（5.4.1节）、基于文件的攻击（5.5.3节）和了解产品的行为，测试人员需要分析产品会使用哪些文件。

例如，测试人员小张使用Process Explorer来分析必应词典桌面版所打开的文件。他在Process Explorer中选择进程BingDict.exe，按下组合键【Ctrl+H】以显示该进程打开的句柄。他浏览如图7-16所示的文件句柄，获得了一些测试想法。

```
File    C:\Program Files (x86)\Microsoft Bing Dictionary
File    C:\Program Files (x86)\Microsoft Bing Dictionary\data\CN\CHS.dat
File    C:\Program Files (x86)\Microsoft Bing Dictionary\data\EN\ENG.dat
File    C:\Users\Liang\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\4K2FYSFI\ad_bdc[1].htm
File    C:\Users\Liang\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\4K2FYSFI\clienthomepage[1].htm
File    C:\Users\Liang\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\4K2FYSFI\ehclinknew[1].htm
File    C:\Users\Liang\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\SAYZ12YC\clientsearch[4].htm
File    C:\Users\Liang\AppData\Local\Microsoft\Windows\Temporary Internet Files\counters.dat
```

图7-16 必应词典的文件句柄（局部）

- ❑ BingDict.exe打开了CHS.dat和ENG.dat。从文件名和文件路径推测，它们是中文词汇和英文词汇的数据文件。小张记录该发现和推测，以便稍后向程序员询问。此外，他还记录一些测试想法：如果数据文件损坏，必应词典如何处理？如果数据文件丢失，必应词典如何处理？安装程序的“修复”功能能否恢复被损坏或被删除的数据文件？

- BingDict.exe打开了IE临时目录中的一些网页。之前的分析显示，必应词典加载IE的动态链接库，所以小张推测必应词典会调用IE来显示这些网页。于是，他用IE来打开这些文件，实验结果表明clienthomepage[1].htm就是必应词典的默认主页（参见图7-17）。这提示小张可以考虑针对网页的测试：如果必应词典收到错误的、甚至恶意的网页，最严重的后果是什么？如果必应词典迟迟不能收到网页，会发生什么？会不会出现只有一部分网页被显示的情况？

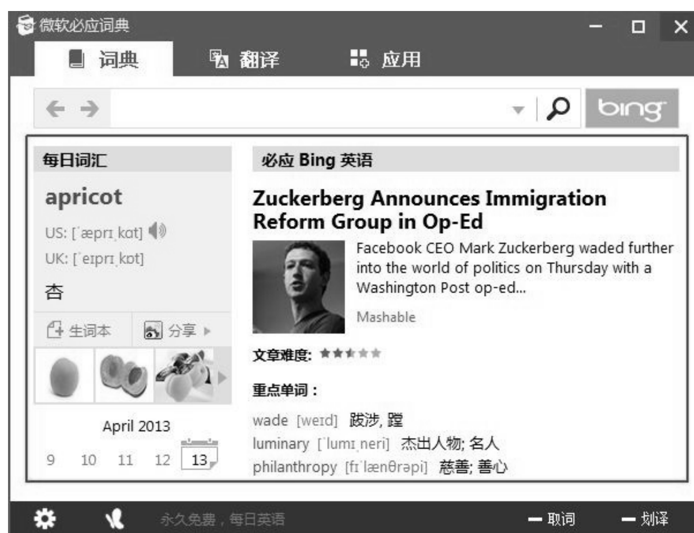


图7-17　矩形区域的内容来自clienthomepage[1].htm

任务3：分析程序之间的协作关系

一些软件系统包含多个程序，它们相互协作来完成任务。为了更好地理解系统，测试人员需要了解组成系统的程序，并分析它们之间的关系。

例如，测试人员小张点击必应词典桌面版的“取词”（该命令位于图7-17的右下角），以启动屏幕取词功能。随后，他在Process Explorer中发现，进程BingDict.exe启动了子进程WordCapture.exe，后者又启动了进程FuncServer_WDC_x64.exe（参见图7-18）。从进程的名字推测，WordCapture.exe负责取词功能，FuncServer_WDC_x64.exe处理64位进程的取词。

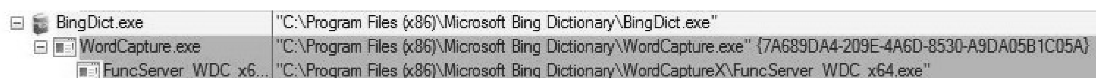


图7-18　必应词典为“取词”启动子进程

然后，小张再次点击“取词”，以关闭屏幕取词功能。他用Process Explorer发现，进程BingDict.exe关闭了进程WordCapture.exe和FuncServer_WDC_x64.exe。为了了解更多的信息，小张启动

Process Monitor①，接着再次执行启动屏幕取词和关闭屏幕取词的操作。之后，他在Process Monitor中按下组合键【Ctrl+T】，以显示进程树对话框。图7-19是该对话框的局部，显示了BingDict.exe及其子进程的生命周期。该图显示了这些进程的开始时间和结束时间，确认了之前的分析，即BingDict.exe在启动屏幕取词时创建子进程，在关闭屏幕取词时结束子进程。



图7-19　Process Monitor显示了进程的生命周期

基于以上分析，小张记录了一组继续调查的想法。

- 向程序员了解BingDict.exe、WordCapture.exe和FuncServer_WDC_x64.exe的通信方法。
- 分析WordCapture.exe和FuncServer_WDC_x64.exe加载的动态链接库。
- 分析WordCapture.exe和FuncServer_WDC_x64.exe打开的文件句柄。
- 测试对64位进程和32位进程的取词。
- 如果WordCapture.exe崩溃，BingDict.exe会如何处理？会重启该进程吗？
- 如果FuncServer_WDC_x64.exe崩溃，WordCapture.exe会如何处理？会重启该进程吗？
- 如果WordCapture.exe陷入死循环，BingDict.exe能感知该情况，并作出合理反应吗？
- 如果FuncServer_WDC_x64.exe陷入死循环，WordCapture.exe能感知该情况，并作出合理反应吗？
- 与“取词”并列的“划译”功能会让BingDict.exe启动子进程吗？
- 必应词典的安装目录下还有没有其他可执行程序？

任务4：分析客户端与服务器之间的通信

目前，许多应用需要访问互联网资源，需要与服务器上的网络服务通信。这些客户端应用可能是网络浏览器中的HTML/JavaScript程序，也可能是独立运行的软件。了解它们与网络服务的通信，将帮助测试人员理解它们的运行机制，并从网络通信的角度产生一些测试想法。

例如，测试人员小张使用Web调试代理Fiddler②分析必应词典桌面版（2.0版）与服务器的通信。他在必应词典的查询框中输入单词“rainbow”，并按下回车键。Fiddler捕获了必应词典的请求和服务器的响应，其结果如图7-20所示。小张发现第一条会话的数据流量较大的，于是选中该会话，获得如图7-21所示的请求和响应的细节。

#	Result	Protocol	Host	URL	Body	Caching	Content-Type	Process
1	200	HTTP	cn.bing.com	/dict/clientsearch?mkt=zh-cn&setLang=match&form=BDVEHC&q=rainbow	11,908	private	text/html; c...	bingdict:10760
2	200	HTTP	cn.bing.com	/fd/ls/l?IG=ba293a22e1fa4cdda3a511b9d4ec4873&CID=0459679482A7...	42	no-cache;...	image/gif	bingdict:10760
4	204	HTTP	www.bing.com	/dict/xls.aspx?Type=Event.ClientInst	0	private		bingdict:10760

图7-20　Fiddler捕获的必应词典与服务器的通信

① http://technet.microsoft.com/es-ar/sysinternals/bb896645。

② http://www.fiddler2.com。



图7-21　Fiddler显示的请求和响应

通过分析图7-21，小张了解了必应词典与服务器通信的协议，并获得了一些测试想法。

- 必应词典向服务器（cn.bing.com/dict/）提交一个Web表单，该表单包含用户查询的单词“rainbow”。服务器的响应是一个HTML页面，该页面是对“rainbow”的解释。由先前的分析可知，必应词典会调用IE的动态链接库来显示该页面。
- 在实现技术上，必应词典2.0版与1.7版有显著区别。图7-22展示了1.7版的请求和响应，其中所查询的单词是software。1.7版发送的请求是一个基于XML文档的SOAP消息，包含被查询的单词。1.7版接收的响应也是一个SOAP消息，包含对单词的解释。必应词典需要解析该XML文档，并利用Windows Form技术来显示单词解释。2.0版在提交请求时用表单取代了SOAP消息，在处理响应时用IE和HTML取代了Windows Form和XML，从而显著地提高了软件的性能。
- 实现技术的变化暗示项目团队希望必应词典有更好的性能，因此需要执行性能测试来检验新设计是否提高了性能。重点测试内容是单词查询的速度，以及必应词典对计算机资源的占用。
- 从测试服务器的角度，如果必应词典发送的请求是错误的，服务器会作何响应？
    - 如果请求缺少一些必要的参数，服务器会如何响应？
    - 如果请求包含额外的参数，服务器会如何响应？
    - 如果请求提交的参数值包含错误（如空值、很长的字符串、非法字符等），服务器会如何响应？

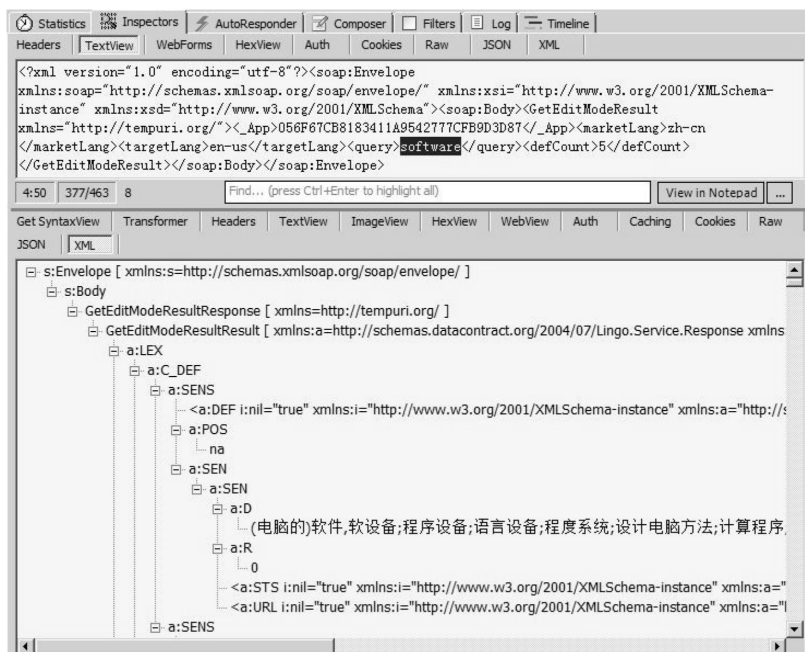


图7-22　必应词典（1.7版）的请求和响应

❑ 从测试必应词典的角度，如果服务器返回的响应是错误的，必应词典会作何处理？

- 如果服务器不返回响应，必应词典的请求超时是多长？超时后，必应词典如何向用户报告错误？
- 如果返回的响应不是合法的HTML页面，必应词典如何处理？
- 必应词典调用IE来显示HTML页面。能否构造一个HTML页面，使得显示结果或显示过程出错？
- 如果黑客劫持了响应，它能否利用跨站点脚本攻击等手段来攻击用户？必应词典有没有对返回的HTML页面进行必要的安全性检查？

任务5：分析产品的性能

在测试过程中，测试人员需要留意产品的性能问题。一旦发现某个操作会导致软件反应缓慢或失去响应，他就应该对该操作进行性能分析。所获得的信息不但有利于策划完整的性能测试，也有助于及时解决性能问题。在许多时候，用户觉得产品“很慢”，并不是因为软件的所有操作都很迟钝，而是一些关键操作反应缓慢。尽早地发现主流场景中的缓慢操作，有利于项目团队制定修复计划，调整软件设计来优化性能。如果性能问题在项目后期才报告，那么项目团队很可能没有足够的时间来实现性能优化。

分析性能的工具很多，测试人员应该根据测试对象、测试任务、期望获得的信息，来选择

合适的测试工具。在Windows平台上，Windows Performance Monitor[①]、Windows Resource Monitor[②]、Windows Performance Toolkit[③]是常用的性能分析工具。在日常测试活动中，测试人员可以利用一些轻量级的工具，做快速的性能分析。例如，我最常用的性能分析工具是跑表和Process Explorer。

在测试过程中，如果我发现一个操作较慢，我会拿出跑表对其计时。跑表简单易用，能够快速记录一个操作的用时。虽然它不适合度量快速的软件运算，但是对于绝大多数用户可感知的延迟，它能够提供足够精确的计时结果。倘若我认为该操作的迟缓是一个性能问题，我会执行操作若干次，将平均用时写入缺陷报告。对于缺陷报告而言，“执行该操作5次，平均用时15.5秒，最长用时17.2秒”比“改操作反应迟钝，用时超过10秒”是更好的问题描述。

除了计时，我还会用Process Explorer监视进程的性能。在Process Explorer中，双击一个进程，就可以打开它的属性对话框。选中属性对话框的性能图标签页，就可以显示该进程的CPU、内存、I/O（输入与输出）的性能数据。我使用Microsoft Word编写本书的文稿，图7-23是Process Explorer记录的Word保存本书文档的性能图。分析该图，我可以得到一些性能相关的事实和推测。

- 在文档保存过程中，Word的CPU使用率的峰值是25%。峰值区域大致可以分成三段，其中第二段的内核用时特别高，说明操作系统在这段时间很忙碌。
- 测试计算机的CPU有4个内核，25%的CPU使用率暗示文档保存操作完全占据了1个内核。因为Word使用UI线程保存文档，所以UI线程在这段时间忙于保存操作，不能响应用户输入。对于用户而言，Word似乎失去了响应，陷于“假死”状态。如果该状态持续几秒的时间，用户就会感到烦躁和不安。
- 在文档保存过程中，Word的内存占用有一定的提升，说明保存文档需要额外的内存来完成任务。文档保存结束后，Word会释放内存，但是内存占用量并没有回到保存之前的水平。通常，这是因为程序使用了缓存技术，一些被缓存的内存并不会被立即释放。不过，这也提示我检查多次保存文件是否会持续提高内存占用量，以致于影响到Word和计算机的性能。
- 在保存过程中，I/O有很大的波动，其峰值出现在CPU峰值的第三段。这暗示CPU峰值的前两段是在准备输出，第三段才是真正地将数据写入硬盘。当CPU峰值结束后，I/O仍有几次较大的波动。这暗示Word调用了Windows的异步输出命令，用较低的CPU使用率完成了余下的输出任务。

在提交性能缺陷时，我会考虑附上类似图7-23的性能图。这有助于更直观地描述软件的性能表现，帮助程序员诊断性能问题。

① http://technet.microsoft.com/en-us/library/cc749249.aspx。

② http://msdn.microsoft.com/en-us/library/windows/desktop/aa372266(v=vs.85).aspx。

③ http://msdn.microsoft.com/en-us/library/windows/desktop/hh162945.aspx。

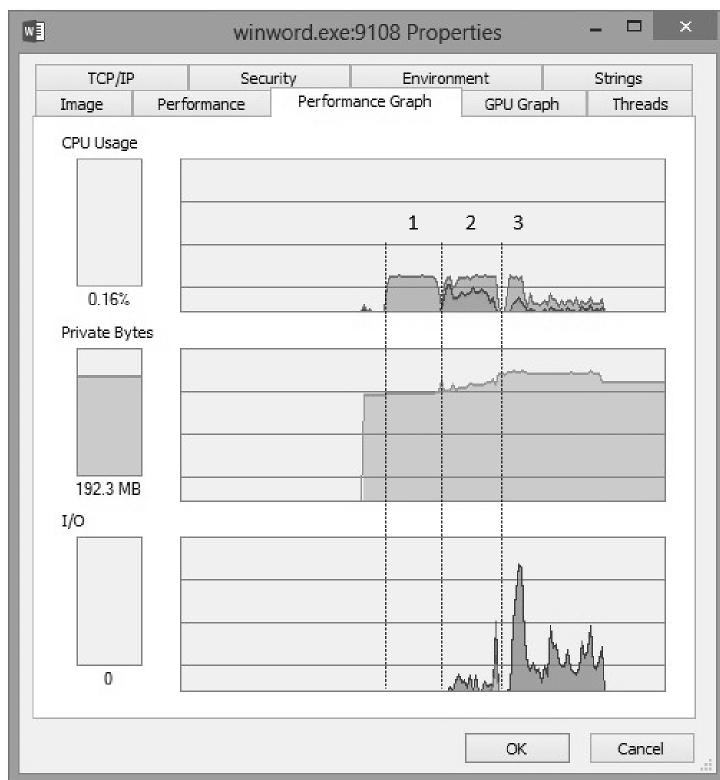


图7-23　Word保存文档的性能图

动态分析产品的方法和工具是多种多样的。本节仅讨论了少数几种实践方法，更多的分析技术和工具有待测试人员在具体的测试活动中探索。由本节的讨论不难看出，合理地使用工具，可以快速地了解产品的设计，提高与程序员交流的效率，并实施更有效地测试。

7.2.2　在调试器中观察软件行为

7.1节讨论了在不运行软件的情况下通过分析源代码来理解软件设计，7.2.1节介绍了在不参考源代码的情况下通过工具来了解软件实现。本节讨论一种同时运用源代码和工具来分析软件的方法——使用调试器观察软件行为。

软件实现往往是复杂的，静态地阅读代码有时不能完全掌握软件的动态行为。利用调试器监控软件的运行，将实际执行的控制流、数据流和源代码结合起来，能够更快地理解设计，有事半功倍之效。对于难以理解的代码，测试人员可以用调试器设置断点，在程序执行命中断点后，用单步执行的方式来观察执行语句序列和变量取值的变化。这样能获得设计的细节信息，为进一步的探索和测试提供信息。

例如，测试人员小张为了深入了解NUnit的约束，决定调试单元测试用例AllItemsAreNotNull。他使用Visual Studio 2012在单元测试的第一条语句设置断点，然后运行测试，详细情况请参考图7-24。随后，他单步执行程序，重点关注表达式Is.Not.Null、构造函数AllItemsConstranit和断言函数Assert.That的实现。

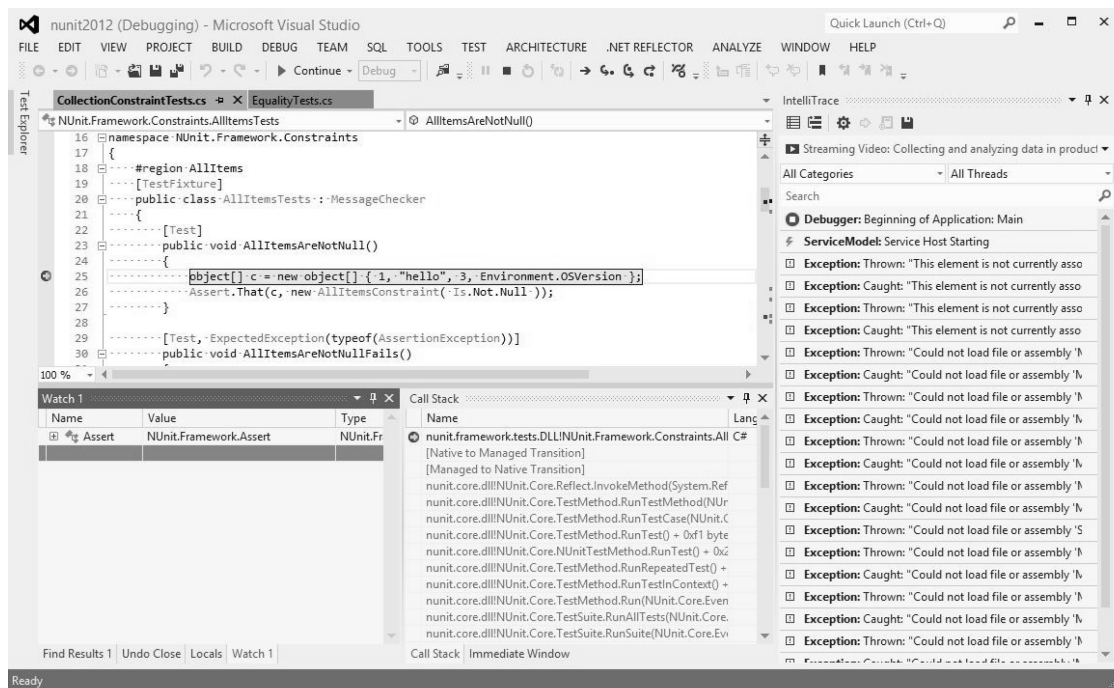


图7-24　使用Visual Studio调试NUnit代码

在单步执行Is.Not.Null的过程中，小张阅读了属性Not、属性Null的实现代码，详情参见图7-25。由这些代码，他了解到Is.Not.Null会返回一个约束表达式对象，该对象包含NotOperator和NullOperator。

在单步执行构造函数AllItemsConstraint的过程中，小张阅读了AllItemsConstraint的父类PrefixConstraint的构造函数和ConstraintExpress.Resolve的实现代码，详情参见图7-26。这些代码说明AllItemsConstraint通过调用函数Resolve将约束表达式对象（Is.Not.Null的返回值）转化为约束对象。Resolve的实现利用操作符栈来归并生成约束对象。该设计类似于惰性初始化，即构造约束表达式Is.Not.Null时并不生成约束对象，直到构造具体的约束对象时，才转换表达式。

```
/// <summary>
/// Returns a ConstraintExpression that negates any
/// following constraint.
/// </summary>
public ConstraintExpression Not
{
    get { return this.Append(new NotOperator()); }
}


/// <summary>
/// Returns a constraint that tests for null
/// </summary>
public NullConstraint Null
{
    get { return (NullConstraint)this.Append(new NullConstraint()); }
}


/// <summary>
/// Appends an operator to the expression and returns the
/// resulting expression itself.
/// </summary>
public ConstraintExpression Append(ConstraintOperator op)
{
    builder.Append(op);
    return (ConstraintExpression)this;
}
```

图7-25　Is.Not.Null的实现代码（摘录）

```
public class AllItemsConstraint : PrefixConstraint
{
    public AllItemsConstraint(Constraint itemConstraint)
        : base( itemConstraint )
    {
        this.DisplayName = "all";
    }


protected PrefixConstraint(IResolveConstraint resolvable) : base(resolvable)
{
    if ( resolvable != null )
        this.baseConstraint = resolvable.Resolve();
}


public Constraint Resolve()
{
    if (!IsResolvable)
        throw new InvalidOperationException("A partial expression may not be resolved");

    while (!ops.Empty)
    {
        ConstraintOperator op = ops.Pop();
        op.Reduce(constraints);
    }

    return constraints.Pop();
}
```

图7-26　AllItemsConstraint的构造函数调用ConstraintExpress.Resolve来获得Constraint对象

在单步执行Assert.That的过程中，小张阅读了函数That和AllItemsConstraint.Matches的实现代码，详情参见图7-27。断言函数Assert.That会调用AllItemsConstraint.Matches来检查传入的可迭代对象（在当前执行的单元测试中该对象是一个数组），而AllItemsConstraint.Matches会使用约束（Is.Not.Null）来检查可迭代对象所以返回的每一个元素。

```
static public void That(object actual, IResolveConstraint expression, string message, params object[] args)
{
    Constraint constraint = expression.Resolve();

    Assert.IncrementAssertCount();
    if (!constraint.Matches(actual))
    {
        MessageWriter writer = new TextMessageWriter(message, args);
        constraint.WriteMessageTo(writer);
        throw new AssertionException(writer.ToString());
    }
}

public override bool Matches(object actual)
{
    this.actual = actual;

    if ( !(actual is IEnumerable) )
        throw new ArgumentException( "The actual value must be an IEnumerable", "actual" );

    foreach(object item in (IEnumerable)actual)
        if (!baseConstraint.Matches(item))
            return false;

    return true;
}
```

图7-27　断言函数Assert.That的实现

经过以上调试步骤，小张了解了约束和断言函数的实现技术。虽然他没有检查所有的约束和断言代码，但是他根据这个实例能够体会整体的设计方案和具体的设计模式。这些知识能帮助他举一反三地理解相关代码，提高未来的静态分析和动态分析的效率。

通过调试来理解代码之所以高效的一个原因是，调试过程使代码分析处于一个具体的语境中。在调试中，测试人员明确地知晓代码的输入值、程序的状态和期望完成的任务。这使得分析过程目标清晰、语义明确、易于操控。此外，调试让处于不同目录和文件的代码按代码执行顺序出现在测试人员眼前，这帮助他在更大的范围内审阅代码，建立对设计的整体印象。

在产品研究过程中，静态分析和动态分析需要相互支持。以下三项活动通常是迭代进行的，并没有固定的执行顺序。一项活动所获得的信息会指引另一项活动进行更深入或更宽泛的调查研究。

- 静态地分析源代码，以理解产品设计，并产生测试想法和调查目标。
- 使用工具分析产品的行为，以研究产品的动态行为，并产生测试想法和调查目标。
- 使用调试器等工具动态地分析源代码，以理解产品设计，并产生测试想法和调查目标。

调试器的另一个常见用途是分析测试执行是否覆盖特定的代码。例如，测试人员需要测试一个产品补丁。为了检查他的测试用例可以覆盖到变更的代码，他在调试器中启动产品，在修改处设置断点。如果断点被命中，他知道测试用例覆盖了代码变更。如果断点没有被命中，他知道测试设计有缺点，需要改进。在测试过程中，他使用调试器来获得测试反馈，从而快速地调整测试

设计，并确保覆盖所有被修改的代码。典型的代码覆盖率分析要求代码插桩和覆盖率分析工具，对于补丁测试等任务可能不够方便。使用调试器可以快速分析少量代码的测试覆盖率，在一些小规模的测试任务中可以发挥好的作用。

7.3 业务研究

语境驱动测试认为，产品是一种解决方案。如果问题没有被解决，产品就是无用的。[ContextDrvienTesting12]。为了更好地测试，测试人员需要理解产品的业务需求（有待解决的问题），并分析产品设计如何满足业务需求（产品所给出的解决方案）。本节介绍一组研究业务、需求、规约和设计的方法，以帮助测试人员更好地实施测试。

7.3.1 理解关系人

关系人是软件的利益相关者，包括软件产品的购买者、使用者、开发者、维护者、销售者、管理者等。表7-2列举了典型的软件关系人和他们对软件质量的期望。

表7-2 典型的软件关系人

关系人类型	描　述	对软件质量的期望
用户	用户是使用软件的人	用户希望软件可以帮助他实现个人目标，重视软件的能力、易用性、魅力、性能等[①]
客户	客户是购买软件的人。对于个人用户，用户就是购买软件的人；对于企业用户，通常是业务主管批准软件购买申请或签订软件开发合同	客户希望软件可以帮助团队实现业务目标。除了用户看重的质量属性外，客户会重视软件的价格、可靠性、售后支持等
IT管理员	在企业中，IT管理员负责部署、升级、配置和维护企业所使用的软件	IT管理员希望软件可以方便地部署与维护，重视软件的易部署性、可监控性、安全性等
第三方开发人员	许多软件提供编程接口，允许其他开发人员在其之上构建应用	开发人员希望软件提供稳定、丰富、易用的编程接口和充分的文档。他们还希望，当软件升级时，编程接口可以向后兼容
程序员	程序员是编写软件的人	程序员希望软件代码简洁、容易理解，设计灵活、可重构，架构清晰、可扩展
测试人员	测试人员是测试软件的人	测试人员希望软件拥有可测试性，能够方便地控制软件的行为、观察它的状态和输出，易于编写自动化测试、易于审计和追踪
管理人员	管理人员是管理软件开发的人	管理人员希望软件可以达成项目团队的业务目标，这通常体现为用合理的成本、恰当的进度，发布足够好的软件

① 不同的人对于软件的质量属性有不同的期望，表7-2只讨论了几种典型的质量属性。有关质量属性的更详细讨论请参考3.2.9节和4.2.1节。

（续）

关系人类型	描　　述	对软件质量的期望
运维人员	对于互联网应用，运维人员维护在线系统	与IT管理人员相似，运维人员重视软件的易部署性、易升级性、可监控性、安全性等
支持人员	支持人员通过电话、论坛、即时通信软件等工具向用户提供软件的技术支持	支持人员希望软件易于使用、稳定可靠，使用户不必寻求技术支持。他们还希望，当软件出现问题时，它能够提供足够多的诊断信息和简单易用的报告方法，以便于调试和修复
销售人员	销售人员负责推销软件	销售人员希望软件有足够的卖点，从而在激烈的竞争中可以脱颖而出，重视软件的能力和魅力

为了让软件产品获得成功，项目团队需要理解软件关系人的类型，设定为他们服务的优先级，使所开发的产品以恰当的优先级满足他们的需求。对于测试人员而言，他需要用关系人的视角考察软件，针对他们的需求提供软件的质量信息。

3.2.8节介绍了The Test Eye发布的“37个测试想法来源”[TheTestEye12]。这篇文档提示测试人员从以下5个方面思考“关系人”，以产生测试想法。

- **用户**：分析使用软件产品的不同类型的用户，了解他们对软件的期望、需要完成的任务、所具备的技能水平、使用软件的语境等信息。这帮助测试人员多角度地考虑产品的使命和任务，并建立有代表性的“用户角色”（参见5.6.2节）。
- **质量特性**：分析产品需要重点支持的质量特性，并拟定测试策略来检查这些特性。
- **产品恐惧**：了解关系人对产品的忧虑，发现他们真正担心的事情。将这些忧虑转换为测试想法，以指导具体的测试活动。
- **使用情景**：分析关系人使用产品的情景，以设计情景测试（参见5.6节）。
- **领域信息**：了解关系人的领域信息，包括他们要解决的业务问题、使用软件的环境、常面临的困难等（参见7.3.5节）。通过访谈、会议、电话、邮件等方式直接交流，能够更好地理解关系人的动机、任务和想法。

在测试过程中，测试人员需要持续思考这些因素。这可以帮助他更好地理解产品，并产生多样化的测试想法。以下是一些实践要点。

- **在项目之初，测试人员并不深刻地理解产品和项目**。此时，分析关系人的目标是大致理解产品的远景和功能，了解最重要的关系人，以及他们最重视的情景和质量特性。基于这些知识，测试人员会获得一批测试想法，为深入测试奠定基础。
- **随着项目进展，测试人员会逐渐掌握更多的产品知识和业务知识**。他将这些知识放入关系人的语境，用关系人的视角推敲产品，从而产生一些新的测试想法。因为这些测试想法参考了关系人的情景和期望，所以能够更加真实地检验产品在用户环境中的表现。
- **在软件发布后，测试人员会陆续收到一些来自用户的缺陷报告**。他需要分析为什么测试

活动没有捕获到这些缺陷，并了解用户发现它们的情景。这些知识有助于他更好地理解产品和用户，以准备未来的测试。

在测试实践中，吃狗食，即大规模内部使用，是一种从软件使用者的角度获取产品质量信息的方法。所谓“吃自己的狗食”是指软件项目团队在日常工作中使用自己开发的产品[WikipediaDogfooding13]。例如，微软的Windows团队会使用开发中的Windows系统。团队成员定期将新版本的系统安装在自己的计算机上，将它作为工作平台，完成编码、测试、协作等所有工作。又例如，谷歌公司的员工会安装最新版的Chrome浏览器，用它网络冲浪，以提供第一时间的质量反馈。在浏览器经历了足够的内部使用后，他们会发布公开版本给外部用户使用[Whittaker12]。

内部使用是一项很有价值的开发实践，它可以在许多方面提高开发过程和软件产品的质量。

- **因为大量员工使用产品，所以内部使用可以发现许多小组测试难以发现的缺陷**。例如，Windows团队有数千名员工，他们在使用Windows系统时，会利用多种多样的硬件设备，安装形形色色的软件，以迥然不同的方式进行工作。这有助于发现一些特殊配置和特殊操作才能暴露的缺陷。
- **因为使用者在利用产品完成实际任务，所以内部使用有助于发现一些真实情景所暴露的缺陷**。现实世界是非常复杂的，用户使用软件的方式和需要处理的数据往往超乎测试人员的想象。大规模的使用让软件经受真实世界的考验，有助于发现一些“极端情况”（有时用户的“常见情景”可能是软件设计的“极端情况”）中所暴露的缺陷。
- **内部使用让整个研发团队参与到质量反馈过程中，让每个人都可以提交缺陷和建议，从而潜移默化地提高了团队的质量意识**。因此，有经验的团队领导会鼓励所有团队成员参与内部使用，并以身作则地使用软件、提交缺陷和建议。
- **内部使用要求员工安装尚未发布的最新版本的软件，这潜在要求软件的大部分构建达到“可用”的标准，不存在频繁崩溃、工作流中断等严重的缺陷**。项目团队应该将该要求视为质量改进的动力，用多种方法来提高所提交代码的质量，并通过持续集成和自动化测试来尽快发现严重的缺陷。
- **内部使用对软件开发过程提出了更高的要求**。项目团队需要持续地构建出足够好的产品，用恰当地方式将软件推送给内部用户，实现故障诊断工具以自动地收集软件崩溃、进程死锁等严重问题，提供方便的渠道让内部用户提交缺陷和建议，建立缺陷分拣流程让用户提交的问题得到调查和修复。为了达成这些目标，项目团队需要付出额外的努力，其成果是更流畅的开发过程、更迅速的质量反馈循环。

由以上讨论可知，内部使用需要整个项目团队的支持，并对开发流程和软件技术提出了相应的要求。在此过程中，测试小组的典型任务包括用自动化测试监控构建的质量、安装并使用软件、提交缺陷、复现他人提交的缺陷、提供更多的质量信息等。从理解用户的角度，测试人员在使用软件的时候，除了关注软件崩溃、功能失常等明显缺陷外，还需要分析软件在用户情景中的表现。

在使用软件完成日常工作时，他会主动使用软件的高级功能和高级设置，并尝试用多种方式来操作软件。经过一段时间的“把玩”，他能够较完整地理解软件的能力，并对软件如何解决用户的问题形成自己的见解。

内部使用的潜在问题是项目团队成员通常不是最终用户，或不能代表所有的用户，所以内部使用仍有可能会错过一些明显的缺陷。例如，微软Office团队内部使用开发中的Office套装，虽然有数千人使用Microsoft Word，但是该过程仍存在以下不足。

- 微软的工作语言是英语，所以员工之间用英文文档和邮件进行交流。内部使用对英语文档、英文输入、英语版Word有较好的覆盖，但是并没有很好地覆盖到多语言文档、多语言输入、加载语言包的Word、非英语版Word等情况。考虑到Office套装支持数十种语言、面向全球市场，单纯的内部使用并不能满足国际化测试的要求。
- 微软员工所编写的大多是商业备忘录、规格说明、设计方案、测试计划等技术性较强、篇幅较短的文档，它们没有覆盖许多重要的文档类型和排版需求。例如，科研工作者撰写的学术论文要求精确的排版，并包含大量的数学符号、数学公式、图、表、交叉引用、参考文献等元素。领域专家编写的学术专著可以视为大部头的学术论文（上千页的专著并不罕见），其篇幅给Word的易用性和性能提出了挑战。
- 许多企业和开发者基于Word提供的API编写了大量的宏应用和Word插件。微软的内部使用不能覆盖这批被广泛使用的应用和插件。

为了弥补内部使用在覆盖率上的不足，许多软件团队会发布外部试用的Beta版本，从而更广泛地收集产品的质量反馈。不过，测试小组并不能完全依赖外部试用来发现软件缺陷。他们应该评估已执行测试和内部使用的覆盖范围，然后针对特定用户角色设计一些专项测试。例如，Word的测试小组可以考虑如下测试活动。

- 测试小组邀请母语不是英语（如中文、日语、西班牙语等）的员工参与为期一天的国际语言测试。测试组织者提供测试指南，让测试参与者使用母语编写一份包含指定内容元素（如表格、图片、艺术字、超链接等）的文档。测试参与者使用母语配合其他语言编写文档，利用各种困难的任务来挑战软件。这样的测试活动可以在项目周期中实施数次，以扩大测试所覆盖的语言和文档类型。
- 测试小组安排测试人员针对特定的用户角色和文档类型进行测试。例如，一名测试人员扮演数学教授，编写一篇数学论文，同时另一名测试人员扮演维基百科的编辑，编写一个名人的词条。测试组织者需要精心设计用户角色和待编写文档，以便测试人员可以用差异化的视角来检查软件。

总之，研究关系人是为了更好地理解软件的使命，通过提供质量信息来提高软件对于关系人的价值。为此，测试人员需要识别重要关系人，用“同理心”去理解他们的目的和需求，并将相关知识运用于测试实践。

7.3.2 评审需求文档

软件需求包括项目团队记录的显式需求和没有被正式记载的隐式需求。测试人员需要关注这两种需求，才能做好测试工作。需求文档通常是显式需求最主要的载体，值得测试人员认真研究。本节介绍一些分析需求文档的方法和技巧。

在许多项目团队，典型的开发流程是产品经理编写需求文档，来概述关系人对产品的需求，并详细定义产品的功能和特性。之后，程序员根据需求文档编写设计文档，测试人员参考需求文档和设计文档编写测试计划。因此，需求文档是测试计划的基本参考资料，需求文档评审是测试设计的重要环节。

为了从需求文档中获得尽可能多的价值，测试人员需要“以输出驱动输入”，即用一组预期的评审结果来指引阅读和分析。带着目标阅读可以调动思维，让测试人员更积极地研究需求文档，从而提高评审效率。对于大多数需求文档，测试人员可以参考如下结果来驱动需求文档评审。

- 需求文档的缺陷。
- 产品的测试模型。
- 测试想法和值得调查的疑问。

驱动力1：评审需求文档需要发现需求文档的错误或不足

测试人员应该一边阅读文档，一边标记和评注所发现的问题。然后，他会与产品经理讨论这些问题，让错误得到修正，让误解得到澄清。如果必要，他会将一些问题提交为文档缺陷。

关于如何批判性地阅读文档，Mortimer J. Adler和Charles Van Doren的经典名著《如何阅读一本书》提供了一些很好的建议[Adler72]。测试人员可以参考“结构—主张—评估”的阅读步骤，来评审需求文档。

批判性阅读的第一步是了解文档的结构和主旨。测试人员快速地浏览文档，获得对文档的整体印象，为深入阅读建立知识结构。对于一篇十几页的文档，快速阅读只需要十分钟的时间，基本态度是“不求甚解”，基本方法是“提纲挈领”。在阅读中，测试人员可以略过大部分内容 ，只关注一些重点对象。

- **摘要**。大部分需求文档在开头综述了要解决的问题、要达成的目标。阅读此类摘要性的文字可以立即了解文档的讨论重点。
- **章节标题**。标题概述了章节的内容，标题的层级关系表达了内容之间的隶属关系。浏览标题让测试人员了解文档论述了什么，以及用什么结构来论述。
- **目标和非目标**。许多文档模板提供了“目标和非目标”一节，让文档撰写者陈述该文档需要解决的问题和不会涉及的问题。阅读该节有助于测试人员了解文档的主旨。
- **图**。许多文档用图描述了模块设计、系统结构、人机界面等重要信息。而且，“一图胜千

言”，图可以用一目了然的方式传达大量的信息。读图能够帮助测试人员用较短时间把握整体设计和必要细节。
- **代码示例**。一些涉及实现细节的需求文档会给出代码片段。阅读这些代码可以快速了解目标代码的设计风格和实现策略。

阅读需要目标，才能避免“过目即忘”。快速阅读的主旨是搜索文档最重要的信息，并整理出它们的关系。在此过程中，测试人员可以用如下问题指导信息搜索。即便快速阅读不能提供完整的答案，它们也为后续的评审提供了指引。

- 这篇文档的目标读者是谁？
- 这篇文档代表了哪些关系人的利益？
- 这篇文档要解决什么问题？要达成什么目标？
- 这篇文档提出了怎样的解决方案？
- 这篇文档有什么样的结构？哪些部分是评审的重点？

经过快速阅读，测试人员已经掌握了文档的结构，知晓哪些部分是重点内容。**在批判性阅读的第二个阶段，测试人员会仔细阅读重点内容，分析文档的论点、论据和主张。其中，阅读的基本技术是组织和总结**。

组织是识别出需求文档陈诉的事实、推断、设计、决定等要点，总结是将获得的信息归纳为多种测试模型。这要求测试人员从需求文档中抽取信息，重新排列，以获得更内聚、更精炼的信息表示。该步骤是一个信息转换的过程，测试人员应该忠实于需求文档。他可以标注出自己的疑问和新想法，但不应该改写原文的信息。

许多测试人员喜欢阅读纸质文档。为了组织文档中的素材，他会一边阅读，一边用手中的笔批注文档。他会用萤光笔标记出重要的文字，用签字笔在段落旁边总结出段落主旨，或批注自己的观点。一些助记符号可以提高标注的速度，例如用惊叹号“！”代表重要的信息，用问号“？”表示有疑问的信息。读完文档后，他扫描标记的文字和自己的笔记，整理出关键信息，并组织成模型。

另一种常见的阅读方法是利用文字处理软件，使用软件的功能来高亮文字和加入批注。有些团队将文档存放在共享的服务器上，文档作者和团队成员都可以读到这些高亮文字和批注。此外，测试人员可以同时启动思维导图软件或笔记软件，让它的窗口与文档窗口并排放置，从而方便地将信息从需求文档搬运到思维导图、功能列表、状态图等模型中。这样，测试人员可以一边阅读文档，一边建立测试模型。

经过重点阅读，测试人员应该能够回答第一阶段所列举的问题。如果还存在疑问，这可能暗示需求文档存在不足，没有讨论基本的问题。另一种可能是测试人员对产品和业务缺少必要的知识，尚不能完整地理解需求文档。对于后一种情况，测试人员可以向产品经理或测试伙伴请教，并补充学习一些产品知识。在某种意义上，评审需求文档也是评估测试人员自身知识的机会，他

应该利用该机会来提高知识储备。

在批判性阅读的第三个阶段，测试人员评价和批评文档的内容。在实际阅读过程中，测试人员完全可以在第二阶段就质疑文档，并指出它的错误或不足。第三个阶段让测试人员站在全局的高度，反思整篇文档，并推敲细节。对于一些比较复杂的软件产品，这样的思考有助于获得更好的理解、发现更深层次的问题。以下是一些常见的思考点。

- **在论证时，文档所提供的证据是否正确**？如果文档所依赖的证据是错误的，那么其推论势必存在问题。测试人员需要质疑需求文档的基本证据。例如，文档陈诉的用户情景是否真实地反映了用户的期望？文档对工作流程所施加的业务规则是必须的吗？文档所依赖的论据是事实，还是文档作者的假设？
- **在论证时，文档所提供的证据是否充分**？考虑不周是需求文档的典型问题。当需求文档遗漏了一些因素，软件设计很可能不能应对真实世界的挑战。测试人员需要尽力思考各种可能性，以及早发现此类问题。例如，文档所提供的用户角色具有代表性吗？有没有重要的用户角色被遗漏？文档所陈述的用户情景覆盖了所有典型情景吗？如果只覆盖了部分情景，那么没有讨论的情景对软件设计有何影响？需求文档给出的软件功能可以处理可预见的用户操作吗？可以处理可能发生的异常吗？
- **在论证时，文档的推理过程正确吗**？对事实的错误解读会导致错误的结论。测试人员需要关注论证过程的逻辑错误。例如，产品经理分析在线系统的日志，获得了用户行为数据，然后建立了用户行为模型，并导出了新的软件设计。对此，测试人要需要检查用户行为数据是否典型、该数据能否代表主流用户的行为、对数据的解读是否恰当、所建立的模型是否正确、模型能否导出另一种不同的设计等问题。
- **文档是否清晰地阐述了用户需求，并明确地描述了软件设计的目标**？论述含糊不清是需求文档另一个典型问题。如果程序员错误地理解了需求，他会设计出有缺陷的软件；如果测试人员错误地理解了需求，他的测试活动将遗漏一些重要的缺陷。测试专家Richard Bender对基于需求的测试有长期的积累，他的文章《歧义评审过程》（*The Ambiguity Review Process*[①]）提出了一个精炼的歧义检查列表，以及一组容易引发歧义的词汇列表[Bender07]。虽然他的歧义词汇列表仅提供英文词汇，但是测试人员很容易将其转换为中文词汇。基于这些评审参考资料，测试人员可以逐渐发展出自己的检查列表和词汇列表，从而实现更高效地检查。
- **文档的所设定的目标、所要解决的问题、所提出的方案是否真正实现了关系人的利益**？具体而言，测试人员可以思考：关系人的利益何在？实现目标可以最大化关系人的价值吗？存在其他有价值的目标吗？需要解决的问题是关系人最关心的吗？所提出的方案能够高效地解决问题吗？是否存在更好的替代方案？当前的方案存在哪些风险？需求文档有没有提出相应的风险缓解方法？

① http://www.benderrbt.com/Ambiguityprocess.pdf。

驱动力2：测试人员通过批判性阅读建立的测试模型

除了发现需求文档的问题，批判性阅读会重新组织原文档的信息，以获得面向测试的产品模型。这不但帮助测试人员更好地理解文档，也为今后的测试提供了支持材料。不同内容的需求文档会得到不同的测试模型，一份需求文档也可能导出多个测试模型。在阅读时，测试人员需要积极地思考何种模型能够有效地组织文档的信息，并动手实践。以下是一些常见的测试模型。

- **功能列表（参见3.2.4节）**。测试人员从需求文档中识别出功能点，将它们组织成功能列表。在此过程中，测试人员会推敲功能点之间的隶属关系，并尝试构造出一份脉络清晰、逻辑合理的层次列表，即将文字信息重构为树形结构的信息框架。该过程不但帮助测试人员梳理文档内容，还为功能测试的测试计划提供了基础素材。
- **质量特性列表（参见3.2.9节）**。测试人员用质量特性列表（如[TheTestEye11]）来检查文档。他识别需求文档明确提到的质量特性（如能力、安全性、兼容性等），作为测试想法的来源。同时，他发掘产品应该具有但是需求文档没有记录的质量特性。有时，产品经理会认为某些特性是“不言自明”的，为了让需求文档简洁明了，他没有予以讨论。但是，程序员可能会忽视这些特性，使得产品设计不能满足他们的要求。有时，产品经理会因为疏忽或缺少经验而遗漏一些重要的质量特性。对于这些情况，测试人员需要提出需求文档应该考虑的质量特性，并提交给项目团队共同讨论。
- **Google ACC（参见3.2.2节）**。在知晓产品的功能和质量特性后，测试人员可以用Google ACC模型来整理信息。他用表格来组织属性、功能、组件这三方面的信息，从而洞察产品特性和产品元素之间的联系。绘制表格的过程是测试计划的过程，表格是测试计划的成果，为进一步的测试设计奠定了基础。
- **启发式测试策略模型HTSM（参见4.2.1节）**。HTSM是一个结构化的、可定制的参考模型，包括产品元素、项目过程、质量标准、测试技术等方面。在信息覆盖上，HTSM是功能列表、质量特性列表和Google ACC的超集。对于复杂的产品，测试人员可以将思维导图形式的HTSM（如图7-28[①]所示）作为信息架构，将需求文档的信息逐步填入思维导图。在阅读过程中，HTSM思维导图好似探索指南，指引测试人员发掘特定方面的信息，最终的成果是一幅多角度描述产品的地图。虽然单纯地阅读文档并不能构建完整的HTSM，但是测试人员仍旧可以用它产生许多好的测试想法。测试人员可以将这些想法加入思维导图，使之成为产品信息和测试想法的混合图。
- **输入与输出模型（参见4.2.2节）**。一些文档会详细描述产品或模块的输入、输出和计算逻辑。对此，测试人员可以建立输入与输出模型，以梳理输入和输出之间的关系。在此过程中，测试人员需要分析文档是否遗漏了重要的输入和输出参数，是否完整地思考了输入检查、异常处理、错误报告等问题。

① 图7-28来源于http://www.ministryoftesting.com/resources/mindmaps/。该网页提供了许多测试相关的思维导图，值得测试人员参考。

- ❑ **系统生态图（参见4.2.3节）**。对于部署在多台计算机或拥有多个进程的软件系统，测试人员可以绘制系统生态图，以描述系统部署的拓扑结构、与外部环境的通信方式和子系统之间的联系。即便某些文档提供了相似的图，从测试视角重新绘制一遍并加入相应的测试想法，对于文档理解和测试计划仍旧大有裨益。
- ❑ **实体关系模型（参见4.2.4节）**。如果理解产品需要掌握一组业务对象的关系，无论产品是否使用数据库存储业务数据，测试人员都可以考虑用实体关系模型来辅助理解。基于实体关系模型，测试人员不但能够获得测试想法，还可以检查模型中实体和关系的一致性，从而发现需求文档的缺陷。
- ❑ **状态机模型（参见4.2.5节）**。如果需求文档描述了产品的工作流、状态变迁、约束规则等信息，测试人员可以建立状态机模型，以描述软件状态、状态变迁以及在特定状态下的约束规则。有时，产品经理用文字、列表等形式描述复杂的业务规则。因为规则非常复杂，测试人员很难完全理解，以致于忽视了其中隐藏的问题。用状态图表达状态、变迁和约束，能够以更直观的方式表达业务规则。这帮助测试人员更好地理解细节，从而发现一些业务设计上的深层次问题。

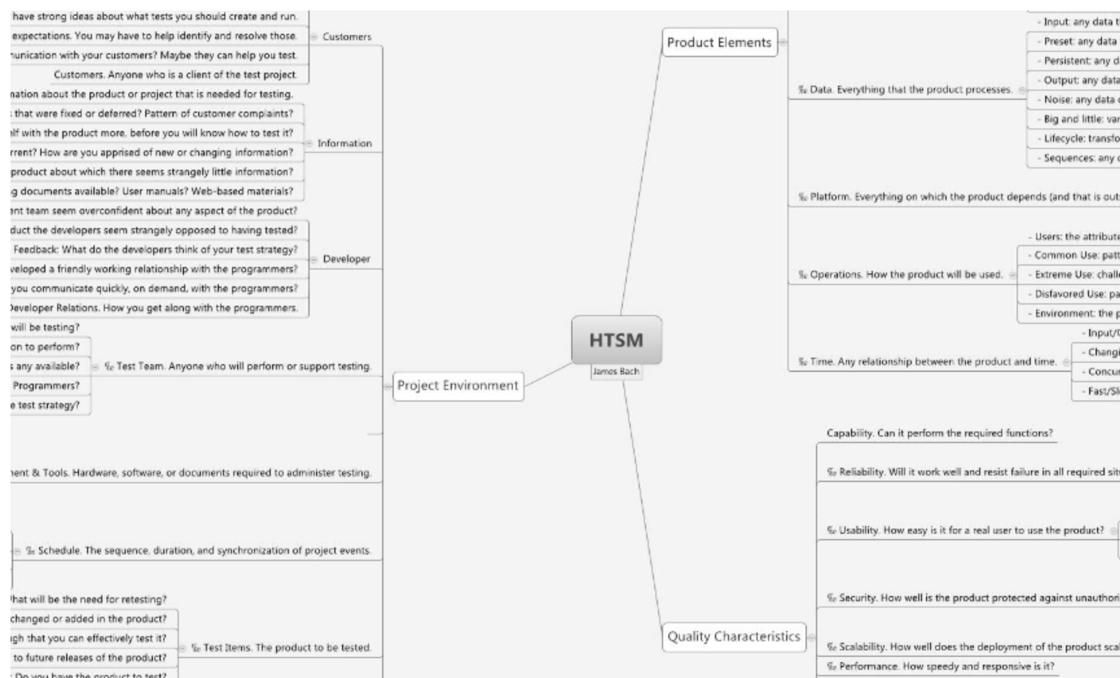


图7-28　思维导图形式的HTSM

驱动力3：测试人员利用需求文档来建立技术调查的目标，并激发测试想法

批判性阅读和构建测试模型都可以产生高质量的测试想法。与此同时，测试人员还可以利用

文档来“核对”产品的行为。在阅读文档时，他标记出文档对产品的陈述，然后针对每一项陈述设计测试。以下是一些常见的测试想法[Kaner11][Hendrickson13]。

- **设计攻击性的测试用例，尝试让软件行为不符合需求文档的陈述**。
- **利用条件分析设计测试用例，以检查产品在不同情景下的表现**。假设，需求文档要求在线词典“在用户键入单词时，显示相应的查询建议”。对此，测试人员可以设计一组测试：如果用户在键入单词时，服务器没有返回相应的查询建议给浏览器，用户会观察到什么？如果用户键入了没有意义的字符串，服务器返回空的查询建议，用户会观察到什么？如果用户键入词组甚至语句，服务器会返回什么结果，用户会观察到什么？
- **改变陈述中的参数值，使用不同的数据、配置、情景来测试产品**。假设需求文档要求软件“能够打开、编辑并保存服务器上的共享文档”。从文档读写的角度，该论述隐含了几个参数：网络带宽、文档尺寸、被编辑的内容、共同编辑文档的人数等。测试人员需要针对每个参数设计测试用例，以测试典型情况、边界情况、异常情况等。
- **测试陈诉所隐含的要求**。例如，“能够打开、编辑并保存服务器上的共享文档”隐式地要求“对共享文档实施权限管理”。这通常意味着软件应该拒绝无权限的用户访问文档，并提供必要信息帮助他申请“只读”或“编辑”权限；软件应该允许拥有“只读”权限的用户读取文档，但禁止他编辑文档；软件应该允许文档所有者授予其他用户权限，并随时撤销授权。测试人员应该考虑这些情景，并设计测试用例。
- **设计测试用例以测试当前陈诉和与之相关的陈述，通过同时覆盖多个模块、功能、情景来发现设计的不一致之处**。
- **设计情景测试，在一个实现用户价值的故事中检查软件是否符合需求文档的陈述**。软件需要交付用户价值，在具体的情景中考查需求陈述和软件行为有助于发现一些需求和设计的缺陷。

在开发和测试的过程中，产品的需求和设计可能发生变化。特别是，在软件投入测试和试用后，项目团队会收到许多用户反馈。为了修复缺陷或优化用户体验，产品经理和程序员常常做出设计变更。随着产品演化，产品设计可能不再符合需求文档的描述。为了避免产品错失了产品研发的“初衷”，测试人员可以在产品基本定型后再次“核对”需求文档和产品。他一边阅读文档，一边按照文档描述来测试产品。其基本目的是检查产品是否满足需求文档的陈述，发现测试遗漏和过时的陈述。对于过时的需求描述，测试人员可以提交文档缺陷，要求产品经理进行修订。另一种注重实效的方法是，测试人员高亮出不再成立的描述，并加上批注：“这段描述不再成立，当前行为请参考软件版本：2.0.1224.1225。”最后将批注后的文档或文档地址发送给项目团队。

在评审需求文档的过程中，测试人员会产生一些测试想法，也会产生一些值得探索的疑问。测试专家Elisabeth Hendrickson建议用测程主题来记录这些测试想法和探索对象，以便为测试活动的组织与安排提供基础 [Hendrickson13]。如3.2.13节所介绍的，测试人员可以参考Hendrickson提供的模板“使用<资源>，探索<对象>，以求发现<信息>”来编写测程主题。在需求评审中，测

试人员可以根据产品和文档的特点来选择模板参数的值。以下是一些常用的选择思路。

- 探索<对象>：
  - 产品的一个功能。
  - 产品的一个组件或子系统。
  - 产品的一项质量特性。
- 使用<资源>：
  - 重要的用户角色。
  - 实现用户价值的使用情景。
  - 挑战<对象>能力的使用情景。
  - 与<对象>有协作关系的功能、组件、外部系统等。
  - <对象>所依赖和处理的数据、资源、配置等。
- 发现<信息>：
  - 产品是否交付了关系人期望的价值？
  - 产品是否实现了需求文档所承诺的质量特性？
  - 产品的行为是否符合需求文档的定义？
  - 产品的行为是否符合测试模型的描述？

在模板的提示下，测试人员可以快速地记录测程主题，例如“使用<最近一个月的产品数据>，探索<产品的计算吞吐量>，以检查<需求文档所要求的吞吐量翻倍是否实现>”、“以<网购买家的身份>，探索<在线商城的退货功能>，以了解<订单的生命周期是否满足现有的状态机模型>”。这再次说明，需求评审的过程也是测试计划的过程，其产出不但是更好的的产品理解，还包括更好的测试计划。

本节针对需求文档介绍了一些文档评审和测试分析的方法。其实，它们也是基于规约的测试的常见方法，能够应用于其他项目文档的评审。在项目过程中，测试人员应该积极收集任何对产品设计和测试有帮助的文档（规格说明、设计文档、用户手册、广告文宣等），通过批判性阅读来逐步完善对产品的理解，并产生的有针对性的测试想法。

7.3.3 通过测试来研究

软件测试是一种研究产品的基本方法。例如，4.2.2节讨论了通过测试来收集产品信息，以构造测试模型，5.4.4节讨论了使用漫游测试来学习产品、调查情况和检测风险。实际上，每当测试人员对产品或业务产生疑问时，他都可以考虑使用测试来实施技术调查，以获得线索和答案。本节介绍一些常见的测试调研方法。

方法1：侦察测程

测试专家James Bach和Jonathan Bach兄弟提出了“侦察测程”的概念。当他们测试一款新产品时，他们会安排一个或几个侦察测程，以快速调查产品的情况。在一个视频中，他们展示了如何快速且多样地实施侦察测程 [Bach09]。其中，测试对象是一个按钮形状的玩具，当按钮被按下时，它会发出“That was easy”的声音。在20分钟的测试过程（视频将整个过程浓缩为4分钟）中，James兄弟使用了一组测试方法。

- 声明测试。他们阅读玩具包装盒上的文字，检查产品的行为是否符合其声明，并尝试从文字中获得测试灵感。
- 文档评审。他们阅读产品帮助，了解产品的正常使用情景和预期行为。
- 功能测试。他们用快乐路径测试检查了产品的基本功能，用状态转换测试检测了按钮按下和弹起触发的状态变迁。
- 产品分析。他们拆卸了产品，分析它有哪些部件、部件有何作用、部件之间如何连接，从而推测产品的工作机制。此外，James Bach提到搜索引擎是很好的测试工具，它可以帮助测试人员快速了解产品信息和相关知识。
- 情景测试。他们构思了一组现实情景来考验产品，如产品从桌子上坠落、产品被一堆杂物压住、产品被液体淋湿、产品被反复锤击等。这些测试对产品具有一定的破坏性，但反应了实际可能遭遇的情况。
- 可用性测试。他们把玩产品，体验其触感，聆听其声音。通过测试结果评估，获得了对产品的第一手印象。
- 压力测试。James Bach使用铁锤强力敲击产品，最终将其外壳击碎。

在测程中，James兄弟使用了多种测试方法，从多个角度侦察产品，获得多个方面的信息。多样化的测试不但较全面地评估了系统，还帮助他们机动地应对新信息，以驱动深入的调查。这提示测试人员应该掌握多样化的测试技术，并勤加练习，以实施灵活高效的测试，从而在短时间内获得更多的测试结果。

不同的研究任务需要不同的研究策略。当测试人员加入一个项目，他需要尽快了解产品的基本情况。因此，其研究策略倾向于运用多种测试技术，以实施广泛的调查。典型的调查成果包括产品的功能列表（梳理产品的功能）、组成结构（了解产品部件及其协作关系）、实现技术（推测产品的工作机制）、输入与输出模型（知晓如何控制和观察软件的行为）、缺陷较多的不稳定区域（分析产品的复杂度和风险）等。当测试人员已经融入项目团队，他的研究任务往往是获得一个新功能或新修复的情报，从而周密地设计测试方案。这时，其研究策略倾向于专注于特定的功能，以实施深入的调查。典型的调查成果包括被测功能的输入和输出、与被测功能协作的其他功能、覆盖被测功能的用户情景、测试想法列表、风险列表等。

方法2：结对测试

除了独立研究，测试人员可以与同事结成测试对子，实施结对测试。所谓结对测试是指两位测试人员并排而坐，使用同一台计算机进行测试 [Kaner07]。其中，一位测试人员充当测试驱动者，设计并执行测试用例，另一位测试人员从事测试辅助活动，包括观察产品细节、提供测试想法、记录测试笔记、查阅参考资料等。他们会定期交换职责，使得两人都有机会驱动测试。

从技术调查的角度，结对测试有助于产生多样化的测试想法。测试驱动者需要向结对伙伴解释他的测试想法，并回答同伴对产品或测试的疑问。该过程能够自然地理清测试思路，并挖掘出更多的测试机会。通过交换职责，来自不同测试人员的经验和技巧自然融入测试过程，扩展了测试的视角和手段。与此同时，结对伙伴之间的密切交流，可以启发彼此的思路，从而激发出更多的测试灵感。

从业务研究的角度，结对测试可以帮助测试人员更快地理解产品。在大型项目中，一位测试人员只负责产品的几个模块，只了解产品的一部分业务知识。为了有效地测试模块之间的交互，测试小组需要测试人员通力协作、彼此辅助。结对测试让测试人员分享他们的知识，并通过测试来说明知识如何应用于测试。这是一种非常有益的团队学习活动。当传授者向伙伴介绍业务知识和产品实现，他会梳理自己的思路，从而激发出更多的测试想法。同时，学习者会提出各种建议和问题，用他的知识和经验来增强当前测试设计。于是，结对测试综合了两人的力量，会发现一些单人测试不能发现的缺陷。

方法3：组队测试

除了结对测试，测试小组还可以考虑组队测试。所谓组队测试是指多位测试人员使用一台计算机，共同测试一款产品。在大型软件中，一个功能很可能与多个功能具有协作关系。有些协作关系很复杂，有些则很隐蔽。为了对该功能实施完整的集成测试，测试小组需要集合团队的力量来群策群力。组队测试是一种有益的集体测试活动，能够弥补测试负责人知识上的不足，发现一两个测试人员难以发现的问题。其常见的实施方法如下。

- ❑ 功能的测试负责人邀请同事参与组队测试会议。典型的受邀对象包括：相关功能的测试人员、熟悉产品的测试老将、对产品不太了解的测试新人等。邀请相关测试人员和测试老将是为了获得更多的产品知识和更丰富的测试想法。邀请测试新人是为了获得“新鲜”的、无拘束的观点，同时也帮助新手尽快熟悉产品。
- ❑ 测试负责人预定一个会议室，准备好两台计算机和投影仪。一台计算机负责执行测试，另一台计算机负责记录测试笔记。投影仪用来展示测试机的屏幕，让参与者都可以看到测试执行的过程。
- ❑ 在会议中，测试负责人执行测试，另一位测试人员记录测试笔记，其他参与者提供测试用例。在测试开始之前，测试负责人介绍被测功能的特性，并鼓励参与者无拘无束地提供测试想法。在测试中，参与者轮流发言，提供测试用例。每提出一个测试用例，测试

负责人都会立即执行它。与此同时，记录员会记录所发现的缺陷或有价值的信息，以避免提出想法和执行测试的循环被信息记录打断。如果参与者对被测功能有疑问，测试负责人应该立即给予解答，以加速测试流程。

- 组队测试可以提出多样化的测试想法。因为不同的测试人员具有不同的知识和技能，所以他们可以设计出丰富多样的测试用例。特别是，相关功能的测试人员可以从新角度设计出集成测试用例或用户情景。这往往会发现测试负责人难以触发的缺陷。为了保证测试想法的多样性，测试负责人要避免某个参与者不停地发言，而打击了其他人的积极性。在执行完一条测试用例后，他可以主动询问一个比较安静的参与者，请他提供想法。通常，安静的参与者已经对问题进行了一段时间的深入思考，能够提高很好的测试用例。
- 组队测试有较好的自适应性。当测试执行发现一个缺陷时，参与者会受到启发，能够围绕问题区域提出一系列测试想法。这有助于深入调查有风险的领域，发掘出更多的缺陷。不过，测试负责人要避免测试设计只局限在一个小范围中。在一段时间的深入测试后，他需要提醒参与者：除了当前的区域，还有哪些重要的区域值得测试。
- 在会后，测试负责人和记录员共同整理测试笔记，并提交所发现的缺陷。然后，测试负责人总结组队测试所获得的知识和经验，将新信息补充到已有的测试模型和测试文档中。最后，测试负责人发送测试报告给测试经理和参与者，总结所发现的缺陷和所获得的知识，并感谢大家的参与和支持。

方法4：头脑风暴会议

另一种与组队测试相似的团队测试活动是头脑风暴会议。为了更好地测试特定对象，测试人员邀请团队成员参加会议，通过积极思考和自由发言，以获得测试想法和潜在风险。与组队测试相似，头脑风暴会议的主旨是汇聚团队的知识，实施更周密的测试。不同点在于，组队测试需要相对稳定的测试对象，因此通常在项目中后期执行，头脑风暴会议一般不执行测试，因此往往在项目前期、测试对象尚未实现时举行。以下是头脑风暴会议的实施要点。

- 功能的测试负责人邀请同事（包括产品经理、程序员等）参加会议。与组队测试相似，参与者的知识和角色应该差异化，以便从多个方面获得测试意见。
- 在会议中，测试负责人主持会议，另一位测试人员记录参与者的建议。设立专职的记录员是因为记录是一项比较繁重的工作，主持人兼职记录会影响他掌控会议过程。
- 在测试开始之前，测试负责人介绍被测功能的特性，并鼓励参与者根据自己的知识和思考提供测试想法和潜在风险。
- 为了获得多样化的想法，测试负责人应该确保每个参与者都有机会发言。一种常见的做法是让参与者轮流发言，每次发言只能提出一个想法。参与者可以放弃当前的发言机会，但不能一次提出多个想法。经过几轮发言，会议可以捕获到大部分重要的测试想法。另一种做法是让参与者独立地写下5个测试想法，每个想法写在一张即时贴上。然后，将所有即时贴粘在在白板上。测试负责人逐一读出即时贴上的想法，并将相似想法的即时贴归并在一起。经过数轮提议，会议可以获得满白板的即时贴和大量的测试想法。

- 在会后，测试负责人和记录员共同整理测试想法。然后，测试负责人归纳总结获得的知识，将新想法写入测试模型和测试文档中。最后，测试负责人向测试经理参与者发送会议总结报告，罗列获得的测试想法和潜在风险，并感谢大家的参与和支持。

除了发现软件缺陷外，任何测试都包含学习和研究的因素。有时，为了更好地掌握产品，测试人员应该有意识地提高研究性活动的比重，主动地探索并记录产品和业务知识。此外，与团队成员协作、集合团队的力量能够更有效率地理解大型产品。

7.3.4 利用互联网资源

互联网的海量信息提供了软件开发的知识库，搜索引擎则是发现知识的重要工具。恰当地利用搜索引擎和网络资源，测试人员可以更好地理解产品并执行测试。以下是一些典型的利用网络资源帮助测试的情景。

情景1：利用搜索引擎查找技术问题的线索

互联网和搜索引擎已经深刻地改变了软件开发的习惯。以往，当技术人员遇到技术困难，他常常要苦苦思索，并尝试不同的解决方法。如今，他可以在互联网上搜索相似的问题，参考他人对问题的分析和解决之道，从而快速地定位问题和解决问题。

例如，测试人员在Windows操作系统上安装产品时，安装程序报告错误，并提供了错误码1306。他在搜索引擎上查询“windows installer error 1306”，发现了微软描述该错误的页面[①]。该页面指出错误1306的根源是目标文件正在被使用以致于不能被更新，一个可能的解决方法是重启计算机。于是，测试人员着手查找被使用的文件以及使用它的进程，并将重启系统作为后备方案。可见，灵活地使用搜索引擎可以快速获得新知，为解决技术问题提供线索。

为了准确地得到问题的线索，测试人员可以参考以下搜索技巧。

- **搜索有“辨识度”的关键词**。为了获得最相关的信息，测试人员应该准确地描述问题，且使用能与其他信息相区别的关键词。对于软件问题，有辨识度的关键词通常来自错误码、错误消息、函数调用栈、所使用的技术（平台、语言、库）等。其中，错误码和错误消息往往是独一无二的，用它们查询可以立即发现相关资料。例如，在C#、Java、Python等语言中，异常的类名和消息是很好的搜索词组；对于Windows的报错对话框，它提供的错误码和错误消息是有帮助的搜索词组。
- **当查询失败时，尝试新的关键词组合**。在许多时候，前几次查询并不能提供答案。此时，测试人员应该考虑重整查询思路，并更换搜索关键词。他可以自问：当前技术问题的外在表现是什么？其独有的特征是什么？然后，写下一两句话，以精要地描述该问题的表

① http://msdn.microsoft.com/en-us/library/windows/desktop/aa372835(v=vs.85).aspx。

象和特征。接着，他从描述中抽取出若干关键词，以进行搜索。如果搜索结果仍不能提供答案，他可以使用关键词的同义词或近义词，再做查询。有时，测试人员需要尝试几种对问题的描述和几套关键词，才能找到准确的答案。

- **使用英文关键词进行搜索**。对于部分技术问题，只有英文资料提供了线索，所以必须使用英文关键词进行查询。这要求测试人员能够用英文描述问题，并了解相应的英文术语。为了储备相应的英语能力，测试人员可以尝试着阅读英文技术文档和英文技术博客。久而久之，他便了解技术领域的术语和惯用表达方法。这不但有助于使用英文进行搜索，还是很好的技术储备方法。

情景2：通过技术支持论坛等工具获得用户反馈

为了向用户提供便捷的技术支持，许多软件企业准备了产品论坛、邮件组、即时通讯软件等沟通渠道。测试小组可以安排测试人员利用这些渠道，收集用户提出的建议，并回答他们的问题。从测试的角度，这样做有多方面的好处。

首先，这有助于培养全员质量意识。测试小组应该安排每一位测试人员（包括测试经理在内）参与技术支持，让所有人与用户接触，了解他们的需求并解决他们的困难。这有助于测试小组切身体会用户的感受，了解产品的优点和不足，从而逐渐培养出团队的质量意识。在具体实施上，测试小组可以安排一位测试人员负责一个沟通渠道，每个月轮换一次。该测试人员每周用5个小时（平均每个工作日1小时）来收集并处理用户反馈，并在月末发送当月的总结报告，概述用户遭遇的问题、提出的建议、使用软件的情景等信息。

第二，分析用户问题可以获得缺陷的第一手资料。在处理一个具体的问题时，测试人员可以按以下步骤进行调查。

- 首先，测试人员需要判断用户提交的问题是不是产品缺陷。有时，所提交的问题并不是软件缺陷引起的。例如，用户不熟悉产品的特性，错误地操作了软件，导致业务流程失败。又例如，产品受到运行环境的限制，没有足够的资源完成用户任务。对于这些情况，测试人员应该向用户解释问题的根源，并提出可能的解决方案。
- 如果问题是软件缺陷引起的，测试人员需要查询缺陷数据库，以判断这是不是一个已知的缺陷。有些缺陷因为难以复现，被解决为“不能复现”。如果用户在他的环境中复现了缺陷，测试人员应该通过邮件、即时通讯软件等工具与用户直接联系，了解他使用产品的环境和情景，为复现并修复缺陷搜集信息。此外，有些缺陷未被修复的原因是研发团队认为它们不够重要，不值得修复。如果用户发现了某个“不被修复”的缺陷，并认为它导致了严重的问题，那么测试人员应该重新激活该缺陷，并报告用户的使用情景和主观感受。他的目标是通过补充更多的信息，让缺陷得到更全面的考虑，从而得到恰当的修复。
- 如果问题是一个未知缺陷引起的，测试人员应该与用户联系，了解他使用产品的环境和

情景，然后尝试着在测试环境中复现该缺陷。无论缺陷是否在测试环境中复现，测试人员都应该提交缺陷报告，并详细阐述已知的信息，包括用户发现缺陷的环境和使用情景、缺陷对用户的影响、为复现缺陷进行的实验及实验结果等。

在以上缺陷调查过程中，测试人员应该主动地与用户联系，感谢他们提交问题，并竭尽所能为他们服务。如果某个缺陷不能在短期内修复，测试人员可以提出一些缓解方案，帮助用户绕开该问题。

此外，技术支持的过程还可以暴露产品在故障诊断上的不足。有些产品在用户许可的情况下，能够自动收集并提交计算机的配置信息、引发故障的调用栈、产品进程的微型内存转储文件等资料。这显著提高了故障诊断和技术支持的效率。然而，许多产品的初始设计并没有考虑故障诊断的需求。对此，测试人员可以分析技术支持的历史资料，结合切身经验，提出故障诊断的需求，并给出每项需求的优先级。这有助于项目团队从运维的角度思考产品，以做出相应的设计。

第三，技术支持活动能够帮助测试人员了解用户使用软件的情景。对于一款被广泛使用的软件，其用户类型具有很大的差异性。不同的用户为了完成其任务，会构思出不同的解决方案，用多种多样的方式使用产品。其中一些使用情景是项目团队在设计产品时没有想到的。所以，阅读用户问题总是能收集到一批“别出心裁”的使用情景。通过分析这些新的用户情景，测试人员可以更好地理解用户的差异性，并为情景测试积累真实的素材。

情景3：多方面地收集用户对产品的评论

随着互联网的快速发展，用户会在形形色色的网站上发表自己对产品的疑问和见解。测试小组可以定期收集并分析各方对产品的评价，从而更好地了解产品在用户情景中表现了哪些优点，又暴露了哪些不足。以下是一些常见的用户评论的信息源。

- 论坛。除了开发方提供的产品论坛，产品的热心用户可能组建独立的论坛，让用户社区自由地讨论各种话题。随着社区的发展，论坛会聚集一批资深用户，他们拥有丰富的产品知识和使用经验，不但能够回答新手提出的问题，还能够为产品发展提供有价值的观点。测试小组可以将此类论坛视为官方论坛的延伸，定期地收集和分析有价值的帖子，以发现产品缺陷、评估产品优缺点、识别用户情景。
- 博客。一些资深用户会开设博客，发表他使用产品的经验和技巧，并点评产品的优缺点。测试人员可以订阅一些质量比较好的博客，坚持阅读，以积累产品知识，并掌握更多使用产品的方法。在测试活动中，这些新知能够激发测试灵感、提高测试的多样性。此外，它们也是“博客漫游”（参见5.4.2节）的基本素材。
- 问答网站。有些用户在遇到产品问题时，会使用问答网站或微博来征求解答。测试人员可以利用此类网站提供的搜索功能，获得用户提出的问题和相应的解答。那些拥有众多评论和“赞同”的问题往往体现了用户的常见需求和疑问。它们是产品设计和测试设计值得参考的信息。

- ❑ 媒体评论。在新产品发布后，一些媒体会测试产品，并发表评测文章或评测视频。这些评论对于产品的推广有潜在、甚至巨大的影响，产品团队需要予以相应的重视。测试人员应该收集相关评论（包括对竞争产品的评论），了解评论者从哪些方面考察软件，用何种方法和工具来具体评价产品的表现。这些信息是“评论者漫游”和“竞争者漫游”（参见5.4.2节）的参考资料，能帮助测试人员从评论者的角度来测试软件，以发现不良设计对软件价值的伤害。在开发过程中发现此类设计缺陷，使研发团队有资源修复问题，从而避免产品发布后立即暴露在猛烈的批评中。

互联网的信息浩如烟海，是无法穷尽的。高效的测试人员会灵活地运用搜索引擎等工具，快速地获取有价值的资料，为测试设计和产品发展提供支持。该探索过程需要以明确的目标作为指引，以便让投入的时间获得切实的回报。

7.3.5　领域研究

通常，软件产品的任务是帮助用户解决特定领域的问题，辅助他们实现业务目标。如果它不能完成领域任务、不能解决领域问题，无论它在其他方面如何优秀，都无法逃脱失败的结局。为了有效地发现此类问题，测试人员需要深入学习业务领域的知识。例如，照片美化软件的测试人员应该学习图像处理的知识，文字处理软件的测试人员应该学习排版知识，财务软件的测试人员应该学习财会知识。这不但有助于发现严重的设计缺陷，还可以发现一些领域专家才能识别的细节问题。

与需求评审相似，测试人员需要用明确的目标来驱动领域研究。否则，他可能花费了大量的时间，却没有获得对测试活动有价值的成果。一般说来，领域研究的目标是获得产品相关的领域模型。该模型可以连接产品团队和领域专家，并指导测试小组实施业务相关的测试。虽然不同的产品拥有迥异的领域模型，但是典型的模型会包含业务领域的基本概念，能够表达产品的业务目标，以及达成目标的技术方案和约束条件。

首先，测试人员需要掌握基本的领域概念和术语。当领域专家相互交谈时，他们会使用大量的术语来表达领域的专属概念，即使用所谓的“行话”来提高交流的效率。在日常工作中，研发团队的成员也应该掌握并使用领域术语。这让产品经理、程序员、测试人员、领域专家等项目关系人能够高效地交流，并为进一步的领域研究奠定基础。

当研究一个新领域时，以维基百科为代表的在线百科全书是很好的起点。测试人员可以从一个最核心的领域术语开始，阅读它的定义，并利用页面上相关术语和参考文献的链接进一步阅读相关资料。因为时间有限，测试人员不能穷尽所有的术语和资料，所以他需要有选择地阅读。

与测程（参见3.2.13节）相似，他可以设定一个时间盒（60到120分钟），在此期间专心致志地阅读在线百科的词条和参考文献。在阅读过程中，他循环执行批判性阅读的基本步骤：结构、

主张、评估（参见7.3.2节）。首先，他扫描词条全文，阅读摘要、标题和图表，以理解术语的基本含义，并掌握全文结构。然后，他直接跳到感兴趣的段落进行阅读，并将有价值的信息摘录到研究笔记中。接着，他评估当前的研究进展，以确定是深入研究该术语对应的概念（阅读词条所列举的参考文献），还是拓宽研究范围（阅读当前词条的相关词条）。随后，他开始对下一篇文章进行批判性阅读。在时间盒内，测试人员应该高速阅读，浏览尽可能多的词条和文献。其目标不是深入理解一两个术语，而是掌握领域的基本情况，为后续的研究奠定宽泛的基础。

在时间盒结束后，测试人员需要整理研究笔记，将原先零散记录的术语、定义和描述，组织成相互联系的概念体系。之后，他可以将笔记分享给领域专家和测试伙伴，并与他们一起讨论研究成果。在讨论中，领域专家和资深同事可以提供更多的信息，帮助测试人员更快地掌握领域知识的要点，并纠正一些误解。

在掌握基本概念之后，测试人员需要了解产品所服务的业务目标。业务目标研究和需求分析很相似，都需要理解用户获得业务成功必须达到的目标，都通过用户访谈、市场调查、文献阅读等手段来收集信息和产生结论。其区别在于，需求分析着重考虑当前产品的功能和范围，研究计算机和软件如何帮助用户；业务目标研究则偏重于业务本身，挖掘用户的使命和任务，而不刻意地思考具体的工作是由产品完成、还是由用户或其他工具完成。

对于测试人员而言，业务目标的研究成果可以是一组用户角色（参见5.6.2节）。一个用户角色是一个具体的人物，但是他（她）刻画了一类用户的共性特征，描述了他们的共同目标。利用用户角色，测试人员可以更好地理解产品的使命，并为情景测试提供基本素材。他针对每个用户角色，进一步列出产品为他（她）提供的核心功能和质量特性，从而将用户的业务目标和产品的能力联系起来。

在知晓业务目标之后，测试人员需要学习达成业务目标的技术方法。例如，照片美化软件的测试人员针对图片的“色彩调节”功能，需要学习一些图像处理知识，包括图片的文件格式、图片的主要属性（尺寸、分辨率、色深等）、图片在内存中的数据格式[①]、色彩编码、色彩变换算法等。在掌握了这些领域知识后，测试人员可以更好地理解产品设计，并设计有针对性的测试用例。从长远看，测试人员通过持续学习，能逐渐成长为领域专家。他的领域知识让他可以有效地测试该领域的大部分软件。

为了学习具体的技术方法，测试人员需要钻研一些领域专著和文献资料。起初，他可以请领域专家或资深同事推荐一些文献。随着研究的深入，他可以根据工作需要和自身兴趣，搜索并学习更多的资料。通常，测试人员不可能脱离工作来进行长时间的学习。因此，注重实效的测试人员会在整个项目过程中利用空闲时间持续学习，根据产品的发展和测试的需求，逐步丰富自己的

① 大多数图像处理软件支持编辑多种格式的图片文件。当读取图片文件时，它会将不同的图片格式转化为一种统一的格式，并保存在内存中。当用户编辑图片时，软件会根据用户命令对内存中的数据进行计算和修改。当用户保存或导出图片时，软件会将内存中的数据编码为用户指定的图片格式，并写入外部存储器。

领域知识。聚沙成塔，集腋成裘，测试人员会慢慢了解领域全貌。

领域研究是为了更好地测试。在学习技术方法的同时，测试人员应该从测试角度收集信息，为测试设计提供支持。以下是一些常见的研究结果。

- **约束规则**。许多领域拥有定律、规则和法条，它们对领域对象产生了强制约束。例如，物体运动应该符合“能量守恒定律”，JEPG图片编码需要遵循国际标准，财会工作必须遵守国家法律法规等。测试人员应该收集领域中的约束规则，将它们作为测试想法和测试先知的参考资料。
- **潜在风险**。在阅读资料时，如果测试人员感觉某些主题艰深难懂，他应该提醒自己，产品经理和程序员在处理该主题时也会感到困惑。如果他们不能很好地应对领域复杂性，其设计和实现很可能包含错误。测试人员应该将复杂性高的主题记录为风险项，并在今后的测试中予以重点关注。除了复杂性，测试人员还需要留意其他挑战产品的因素。例如，严谨的学术文献在讨论一种技术时，不但会介绍它的优点，还会中肯地讨论其缺点。当项目团队选择一个特定的技术，它的已知缺点应该被记录为风险项，在开发和测试过程中予以监控。
- **测试模型**。与需求评审相似，测试人员应该重新组织他学到的知识，形成面向测试的模型。好的模型能够精炼地表达领域本质，为测试设计和测试执行提供有力支持。

综上所述，领域研究要求测试人员持续付出时间和努力，并将研究成果应用于测试实践。其回报也是丰厚的。掌握领域知识的测试人员能够自如地与产品关系人（领域专家、软件用户、程序员等）讨论产品和业务，尽早发现需求分析和软件设计中的缺陷，并设计出更具威力的测试，发现简单测试不能暴露的问题。

7.4 研究策略

本章已经介绍了静态分析、动态分析、关系人研究、需求评审、测试调查、网络调研、领域研究等研究产品的方法。在实践这些方法的过程中，有两个要点值得测试人员注意。

第一，测试人员需要综合运用多种研究方法，才能全面地理解产品。不同的方法从不同的角度调查产品，它们所提供的信息相互补充，而非彼此替代。在测试活动中，有经验的测试人员会交替使用多种研究方法，让它们彼此支持、相互驱动，从而获得深入的调查结果。例如，他浏览业务相关文献，获得了一组测试想法。为了检验想法的有效性，他阅读产品的源代码，以过滤掉无效的想法，并将有效的想法细化为一组测试用例。在测试执行时，他使用调试器等工具分析产品的动态行为。在工具的帮助下，他发现了一些产品缺陷，并对产品设计产生了新的疑问。为了解除困惑，他又去查阅需求文档和领域资料，以求获得更多的线索。

第二，产品研究会贯穿整个项目过程。在软件项目中，并没有泾渭分明的“研究调查阶段”

和“实践应用阶段”，产品研究和测试实践总是并行实施。产品演化会提出新的测试任务，新的任务将驱动新的研究活动，新的研究成果会指导新一轮的测试，而新的测试结果提供了更多的研究素材。高效的测试人员会充分利用测试的迭代过程，以测试任务为指引，通过注重实效的研究，逐步积累产品和领域知识。他知道掌握知识的根本方法是并发地实施测试相关学习、测试设计、测试执行和测试评估，让它们成为相互支持的活动。理论和想法需要进过实践的考验和修正才能真正发挥作用。

7.5　小结

本章讨论了如何从测试的角度来研究产品和业务，介绍了一些实践方法。

- 静态分析通过阅读产品代码，来理解产品实现和代码变更。
- 对于复杂产品的理解，静态分析的典型策略是概览源代码树，了解目录结构、模块划分、代码分布等基本情况，然后细读一部分重要代码来以点带面。
- 影响分析和传播分析是审阅局部代码并产生测试想法的常见技术。
- 黑盒测试并不是基于无知的测试。
- 动态分析通过运行软件来研究产品的关键情景和重要设计。
- 灵活使用包括调试器在内的动态分析工具将显著提高测试效率。
- 测试人员需要识别重要关系人，理解其目标、使命和任务。
- 内部使用是一项很有价值的开发实践，但它不能覆盖所有用户类型。
- 测试人员需要批判性地阅读产品相关文档。典型的文档评审结果包括：文档缺陷、测试模型、测试想法和需要调查的问题。
- 软件测试本身就是一种研究产品的方法。侦察测程、结对测试、组队测试、头脑风暴会议能够帮助测试人员更好地了解产品。
- 互联网提供了海量的测试信息，搜索引擎是发现知识的重要工具。
- 测试人员需要学习领域概念、业务术语、业务目标和技术方法，以实施更深入的测试。
- 综合运用多种研究方法，才能全面地理解产品。
- 测试相关学习、测试设计、测试执行和测试评估是相互支持的并发活动。

第8章

研究项目

在启发式测试策略模型HTSM（4.2.1节）中，产品元素、项目环境、质量标准和测试技术是测试人员思考测试策略的基本要素。本书之前的章节讨论了测试技术、质量标准和产品元素，本章将讨论项目环境。

软件项目是一个内涵丰富的主题，相关著作可谓汗牛充栋。基于本书的主旨，本章只从测试人员的角度来考察软件项目，目标是为测试活动提供更多的信息。具体内容包括项目团队、项目分析和基于风险的测试。

8.1 项目团队

软件项目的主体是项目团队，他们创造了令人赞叹的功能，也引入了让人恼怒的缺陷。团队的人员构成、组织结构、运作机制将深远地影响软件的质量。语境驱动测试认为“人，团队协作，是项目语境中最重要的部分”[ContextDrvienTesting12]。因此，测试人员需要了解项目团队的成员，并知晓团队的运作方式。只有如此，测试人员才能有效地运用测试资源，为团队提供高质量的测试服务。

8.1.1 了解团队组织

在加入一个新团队后，测试人员应该主动了解项目团队的使命、目标和运作方式。这将帮助他理解自己的职责，从而更好地执行测试任务。以下是一些值得思考的问题。

- 大型项目往往由多个团队共同完成，测试人员需要掌握团队之间的协作关系。
  - 测试人员所处的项目团队负责哪些工作？
  - 当前团队有哪些伙伴团队？哪些是上游团队？哪些是下游团队？
  - 当前团队与伙伴团队有哪些具体的依赖关系？
  - 相互依赖的团队如何协同工作？他们对于合作的内容和方式有哪些共识？
  - 对于测试人员而言，他需要为哪些团队提供哪些服务？其他团队又会通过什么方式向他提供支持？

- 测试人员应该掌握团队的人员组成和角色职责。

  - 对于当前团队，产品经理小组、编程小组、测试小组各负什么责任？落实到个人，产品经理、程序员、测试人员各负什么责任？
  - 在当前团队中，产品经理、程序员、测试人员如何协同工作？他们对于合作的内容和方式有哪些共识？产品经理对测试人员的测试服务有何期望？程序员对测试人员的测试服务有何期望？
  - 如果工作中出现了意见分歧，通过哪些渠道可以解决分歧，或形成决议？
  - 在团队中，哪些人是资深员工？哪些人会指导或驱动哪些具体的工作？在工作中遇到困惑，可以向谁请教？

- 测试人员需要了解项目的组织流程。

  - 项目负责人、产品经理、开发经理、测试经理如何协作？他们如何决定项目流程和规则？他们如何评估项目进度并拟定项目计划？在此过程中，测试人员可以提供哪些信息和建议？
  - 项目团队如何定义“工作完成”？其中，“测试完成”的定义是什么？
  - 项目团队如何决定产品可以发布？其中，测试人员需要为发布决策提供哪些信息？

- 测试人员应该知晓测试小组的组织流程。

  - 在测试小组内部，测试人员如何协作？测试经理如何协调测试人员之间的工作？
  - 测试经理如何管理具体的测试工作？他如何了解工作项的内容和完成情况？他会对测试工作作出哪些决策？
  - 测试经理对测试人员有何期望？他希望测试人员以何种方式来汇报工作？他如何设定测试任务的优先级和质量标准？他如何评估测试人员的工作绩效？
  - 哪些人是资深测试人员？哪些人会指导或驱动哪些具体的测试工作？在工作中遇到困惑，可以向谁请教？

为了更好地理解项目环境，测试人员应该主动寻找以上问题的答案。与软件测试相似，他应该从多个渠道收集信息。

首先，他应该向测试经理询问团队组织的问题。他可以和测试经理约定一个时间，一起讨论测试管理和团队协作的问题。更好的做法是约定一个周期性的一对一会议，以便定期提出问题并交换意见。测试经理通常是测试人员的直属领导，负责测试人员的绩效考评。了解他对测试工作的要求和期望，能够帮助测试人员更有效地工作。而且，测试经理是测试小组的领导者，会组织团队活动、分派测试任务、协调测试人员工作。测试人员应该主动询问测试经理如何领导团队，从而更好地在团队中工作。此外，测试经理能够更多地接触高层经理和其他团队的代表，对项目运作有更深刻的认识，能够向测试人员传递许多有价值的信息。

其次，他可以在日常工作中向团队成员请教。测试经理从管理层的角度给出了意见，而团队成员能从实际工作完成者的角度提供信息。他们的知识大多来自测试实践，既有成功的经验，又有失败的教训。在具体的工作上，咨询他们的意见并妥善思考，能达到事半功倍的效果。测试人员应该对所有同事保持谦虚的态度。在团队中，没有人知道所有细节，集合多人的观点才能形成较完整的认识。

最重要的是，他应该通过工作来获得答案。“纸上得来终觉浅，绝知此事要躬行”，许多道理需要结合具体任务，通过动手实践才能真正掌握。测试人员应该利用当前的测试任务来理解项目和团队，并通过反思来形成自己的认识。

例如，测试经理为新加入团队的小张安排了一个测试任务：与另一个团队共同完成系统的集成测试。为了理解任务的背景，小张会向测试经理询问一些具体问题，例如：

- 谁是集成测试的总负责人？谁驱动和协调具体的工作？
- 伙伴团队中，谁负责集成测试？工作分工是什么？
- 在我们的团队中，还有谁参加集成测试？工作分工是什么？
- 集成测试预期何时完成？集成测试的结束标准是什么？
- 如果有问题（例如严重的缺陷、缺少必要的硬件、某位同事休假等）阻碍了集成测试的进展，应该如何处置？
- 您对集成测试有什么期望？
- 您觉得集成测试存在哪些风险？如何发现风险并消除其危害？
- 您觉得哪些测试策略会有帮助？您觉得可能会发现哪些缺陷？

除了与测试经理交流，小张还会咨询参与过集成测试的员工，从他们那里了解集成测试的情况。通常，测试经理会给出一些战略层面的建议，测试人员会给出一些具体战术。根据所收集的信息，小张会编写一份测试计划，发送给测试经理、本团队和伙伴团队的测试人员，以征求改进意见。随后，他着手测试，并根据其他人对测试计划的反馈，对测试活动进行必要的调整。

在测试完成后，小张回顾测试过程，对测试组织和测试设计进行反思。其中，他会再次思考曾经询问过的问题，并形成自己的见解。经过几次测试任务，小张会形成自己的知识体系。当有新人向他咨询测试工作时，他可以给出独到的建议。

8.1.2　语境独立的启发式问题

上一小节讨论了一些面向团队结构和项目组织的调查问题，本节将介绍测试专家Michael Bolton和JamesBach所提出的一组语境独立的启发式问题 [Bolton10]。所谓“语境独立”是指这些问题不局限于特定的软件项目；所谓“启发式”是指它们有助于挖掘信息，但需要测试人员批判性地思考。当测试人员加入新项目或重新评估现有项目时，这组问题可以帮助他调查当前语境，更好地理解自己的工作。

测试专家Ajay Blamurugadas对这组问题进行了归纳整理，绘制了一幅思维导图，详见图8-1 [Blamurugadas13]。该图将Michael Bolton的几十个问题归纳为十余个要点，帮助测试人员多角度地思考软件项目。当测试人员加入新团队时，他可以向测试经理和项目领导询问这些问题，也可以用它们去指导项目研究。无论如何，调查这些问题都可以让他更好地理解项目，为有效的测试提供线索。

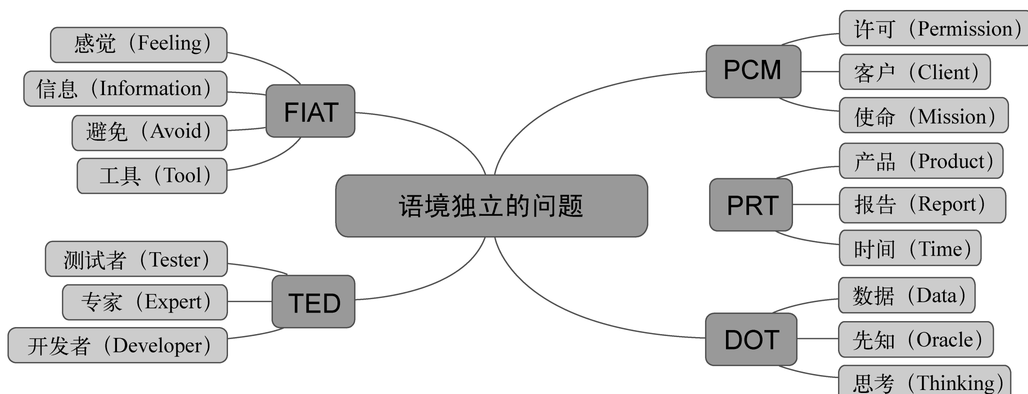


图8-1 Ajay Blamurugadas归纳了Michael Bolton的语境独立问题

具体的问题列表如下。

- 许可（**P**ermission ）
  - 我可以问您一些问题吗？
- 客户（**C**lient ）
  - 谁是我的客户？
  - 您是我唯一的客户吗？
  - 谁是产品的用户？
  - 还有哪些关系人？
- 使命（**M**ission ）
  - 我的使命是什么？
  - 我的使命还包含哪些部分？
  - 据您所知，有哪些问题会威胁产品的价值？
- 产品（**P**roduct ）
  - 我可以无限制地访问产品吗？
  - 是否存在其他相似的产品？

- 被测产品在哪些地方应该与其他产品一致？在哪些地方应该不同？
- 被测产品依赖什么？
- 构建该产品使用什么工具和资源？

❑ 报告（**R**eport）

- 您希望我如何提供报告？多久报告一次？
- 您何时会考虑发布产品？
- 除此之外，您还希望我提供什么？
- 您希望我以何种方式提供？

❑ 时间（**T**ime）

- 我拥有多少时间？
- 距离下次发布或部署还有多少时间？
- 距离测试结束或开发结束还有多少时间？
- 您何时需要报告或答案？

❑ 数据（**D**ata）

- 产品处理什么样的数据？
- 我可以获得一些样例数据吗？
- 我能获得对数据的详细描述吗？

❑ 先知（**O**racle）

- 我如何识别一个问题？
- 如果我认为我发现了一个问题。为什么我认为这是一个问题？还有谁会认为这是一个问题？

❑ 思考（**T**hinking）

- 这是我发现的情况和我对它的解释。还存在其他解释吗？
- 这是我现在的思考结果。还存在其他正确的解释吗？如果我的思考结果是错误的，正确的是什么？
- 您能解释产品是如何工作的吗？
- 您能画一幅草图来解释产品如何工作吗？

❑ 测试者（**T**ester）

- 其他人测试过产品吗？
- 我可以获得他们的测试结果吗？

- 测试小组还有哪些测试人员？
- 我需要具有哪些技术和能力？
- 测试小组还具有哪些其他的技术和能力？何处能提供更多的技术和能力？
- 测试小组可能缺少哪些技术和能力？

❑ 专家（**E**xpert）

- 还有谁了解产品？
- 谁是回答产品问题的最佳人选？
- 在项目团队中，谁是领域专家？
- 谁是公认的领域专家，即便他们在别处工作？

❑ 开发者（**D**eveloper）

- 谁构建了产品？
- 我可以与他们交谈吗？
- 他们容易交流吗？他们乐于助人吗？
- 他们曾经构建过相似的产品吗？
- 存在某个我应该避免与他打交道的人吗？

❑ 感觉（**F**eeling）

- 我对产品和测试有何感觉？
- 您对产品和测试有何感觉？
- 谁可能有不同的感觉?
- 他们可能有何感觉?
- 用户对产品如何评价?
- 我可以与技术支持人员交谈吗?

❑ 信息（**I**nformation）

- 我可以获得哪些信息？
- 还有更多的信息吗？
- 在哪里我可以获得更多的信息？还有其他信息源吗？
- 还有其他形式的信息吗？
- 这就是全部的信息吗？还有其他的规则、需求和规约吗？
- 我可以做哪些工作去帮助您获得信息？

❑ 避免（**A**void）

- 我可以相信谁？谁的信任度比较低？

- 存在一些我不应该做的事情吗？
- 存在一些我应该问您的问题吗？

❑ 工具（Tool）

- 有什么设备和工具可以帮助我测试？
- 您期望我构建什么工具？要求我构建什么工具？

在本质上，以上问题的动机是揭示语境，即通过调查具体问题来更全面地了解项目环境和团队组织。询问并研究这些问题将帮助测试人员更好地体会他的环境和使命。

8.1.3　了解团队成员

人是项目语境中最重要的部分，测试人员需要了解与他一起工作的同事，才能有效地协作、自如地发挥。所谓了解同事，并不是要知晓他们的生活细节，更不是要窥探他人隐私，而是了解他们在工作中如何思考和行动，从而更好地与他们配合。

研究人类性格和思维模式的方法有很多。其中，迈尔斯–布里格斯性格分类法（Myers-Briggs Type Indicator，MBTI）是一种著名的性格测试方法 [WikipediaMBTI13]。测试人员可以借用MBTI的4个基本问题，来分析同事的工作风格，从而以恰当的方式与之合作。

问题1：在使用心理能量时，他是“外向”还是“内向”？

❑ 外向型的人偏向于从外部获得能量，从讨论和互动中获得力量。在工作中，他倾向于与同事密切交流，频繁讨论。所以，测试人员可以考虑和外向型同事当面交谈，共同解决问题。一方面，向他们提供能量和帮助；另一方面，也从他们那里获得力量和灵感。

❑ 内向型的人偏向于从内部获得能量，从思考和冥想中获得力量。在工作中，他倾向于独立工作，安静思考。这不意味着他不喜欢交流，而是他希望在深思熟虑之后再进行讨论。与内向型的同事协作时，测试人员应该用“信任并检验”的风格。首先，相信他们会利用安静的时间独自解决问题，然后，定期与他们面对面交流，同步彼此的进度，讨论一些需要协作才能解决的问题。

问题2：在认识外在世界时，他是依赖 “感觉”还是“直觉”？

❑ 感觉型的人偏向于“眼见为实、耳听为凭”，更多地用五官来认识世界。在工作中，他们重视事实和数据，会深入地研究问题的细节。所以，测试人员在交流之前应该准备好数据、图表和事实，用翔实的信息来推动讨论。

❑ 直觉型的人偏向于使用直觉，用大脑中的非显性知识来分析具体问题。在工作中，他们更重视可以解释事实的观点，用抽象的理论来掌握全局。与他们协作时，测试人员不但

要收集事实，还要思考它们背后的理论，并给出自己的见解。这有助于推动意见交流，在相互启发的过程中获得更深刻的观点。

问题3：在作出决定时，他利用“思考”还是“感觉”？

- 思考型的人在作决策时，重视合理性、逻辑性、因果关系、一致性，偏向于通过理性思考来得出结论。在说服他们时，测试人员需要严密地论证，用坚实的推理来构建证据链，从而让结论显得自然而合理。
- 感觉型的人在作决策时，重视问题的情景，考虑相关人员的感受，并关注解决方案是否协调一致和拥有共识。在说服他们时，测试人员不但要使论证“合理”，还要让结果“合情”。这要求测试人员了解他们的价值观，用“同理心”来体会他们对于问题的感受，从而给出恰当的解决方案。

问题4：在处事风格上，他是“决断”还是“体察”？

- 决断型的人偏向于“快刀斩乱麻”，快速作出决定，让解空间收敛。在软件项目中，决断型的同事倾向于快速给出技术方案，持续推动软件高速发展。这帮助项目团队在激烈的市场竞争中争取先发优势，但也引入了错误决策的风险。为此，测试人员需要慎重思考他们的决定，发掘其中的问题和风险。其目标是确保项目团队对这些问题进行了足够的研究，并为相关风险制定了监控和缓解方案。
- 体察型的人偏向于“循序渐进”，将重大决策延后，保持解空间的开放性，等到拥有足够的信息再作决定。在软件项目中，体察型的同事倾向于只做必要的设计，力求让当前的、真实的问题来驱动软件发展。这有助于保持软件设计的简洁，避免过度设计带来的开销。其风险是他们的决策可能缺乏前瞻性，不能高速地切入新领域或提出新方案，从而错失大幅度提高竞争力的机会。为此，测试人员需要密切关注项目和市场的发展，在必要时提出自己的意见，以推动项目发展。

在运用MBTI的4个问题时，需要牢记：这些问题都是启发式问题，它们可以提供有参考价值的信息，但是无法承诺完美的答案。首先，人的性格是复杂的，不会位于“性格光谱”的两端（例如纯粹的外向或纯粹的内向），而是位于中间的某一点，从而体现出一种混合的风格。其次，人对于不同的事物有不同的态度，从而在不同工作项上表现出不同的工作风格。所以，测试人员除了分析同事的工作风格，还要与他们直接讨论如何协作。有时候，主动问一句：“在这件事上，你希望我做些什么？”，就可以展开很好的对话，为良好的协作奠定基础。

健康的团队需要不同风格的人才，人员在能力和性格上的差异性是一种团队财富。为了在团队中作出更多的贡献，测试人员需要主动了解风格迥异的同事，并采用相应的协作方法。这不但会促成更好的项目团队，也会让自己的职业发展得到他人的支持，毕竟一个善于协作的人总是能获得更多的资源和帮助的。

8.2　面向测试的项目分析

在软件项目中，项目团队会实施多种分析和度量方法，来评估产品质量和监控开发过程。测试人员也可以建立一些度量，来了解项目情况，从而为测试设计提供更多的信息。为了获得较全面的认知，他应该周期性地分析多个项目元素，将片段的信息汇聚成完整的图景。为此，本节将从测试设计的角度，讨论几种最常见的分析对象：软件缺陷、源代码、构建和自动化测试。

8.2.1　软件缺陷

2.6节建议测试人员将缺陷管理系统视作项目的知识库，坚持每天或每周阅读他人提交的缺陷报告。在阅读时，他可以使用一些度量方法来分析近期提交的缺陷，以一览缺陷概况，并为深入研究提供线索。

为了提高度量的效率，测试人员应该尽可能用自动化的方法来获得度量结果。因为大多数缺陷管理系统使用关系型数据库来存储缺陷报告，测试人员可以使用SQL（本节使用Microsoft SQL Server支持的T-SQL）来分析缺陷数据。他只需要向系统管理员申请到数据库的“只读”权限，就可以开始分析。代码清单8-1展现了一个记录缺陷报告的数据表。虽然真实的缺陷表比它复杂许多，但是它呈现了缺陷的基本字段，能够支持一些最常用的查询。

代码清单8-1　缺陷数据表

```
CREATE TABLE [dbo].[Bug](
    [ID] [bigint] NOT NULL,
    [Priority] [smallint] NULL,
    [Severity] [smallint] NULL,
    [Path] [nvarchar](200) NOT NULL,
    [Title] [nvarchar](1000) NOT NULL,
    [OpenBuild] [nvarchar](20) NOT NULL,
    [OpenDate] [datetime] NOT NULL,
    [OpenBy] [nvarchar](10) NOT NULL,
    [ResolveDate] [datetime] NULL,
    [ResolvBy] [nvarchar](10) NULL,
    [Resolution] [nvarchar](20) NULL,
    [FixBuild] [nvarchar](20) NULL,
    [CloseDate] [datetime] NULL,
    [CloseBy] [nvarchar](10) NULL,
    [AssignTo] [nvarchar](10) NULL
) ON [PRIMARY]
```

度量1：统计缺陷提交、解决和关闭的数目

测试人员可以用代码清单8-2所示的查询来获得近1个月以来每日提交的缺陷数。这条语句用`WHERE OpenDate >= DATEADD(dd, -30, GETDATE())`筛选出最近30天的缺陷报告，然后用`GROUP BY CONVERT(date, OpenDate)`对缺陷报告按日期分组，最后用`SELECT COUNT(*) AS BugCount`统计出每

天的缺陷报告数。图8-2展示了相应的查询结果[①]，它表明测试小组在持续发现缺陷，而且缺陷发现的速度比较平稳。

代码清单8-2　查询近1个月以来每日提交的缺陷数

```
SELECT CONVERT(date, OpenDate) AS OpenDay, COUNT(*) AS BugCount
FROM dbo.Bug
WHERE OpenDate >= DATEADD(dd, -30, GETDATE())
GROUP BY CONVERT(date, OpenDate)
```

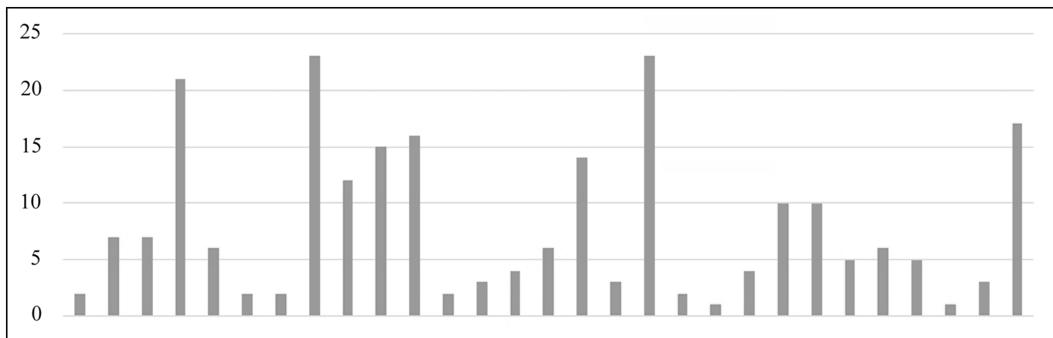


图8-2　近1个月以来每日提交的缺陷数

除了分析OpenDate（缺陷提交日期），测试人员还可以采用与代码清单8-2相似的查询来分析ResolveDate（缺陷解决日期）和CloseDate（缺陷关闭日期——通常是测试人员检查缺陷修复的日期）。这样可以了解解决缺陷的速度和检查修复的速度。

他还可以运行代码清单8-3，来查询近1个月以来的缺陷提交数、解决数和关闭数。如果缺陷解决数远小于缺陷提交数，那么编程小组可能没有足够的时间来修复缺陷。如果缺陷关闭数远小于缺陷修复数，那么测试小组可能没有足够的时间来检查缺陷修复。这都是项目风险，需要及时向测试经理汇报，以便项目管理者作出必要的调整。

代码清单8-3　近1个月以来的缺陷提交数、解决数和关闭数

```
SELECT
    COUNT(*) AS OpenCount
    , SUM(CASE WHEN ResolveDate IS NULL THEN 0 ELSE 1 END) AS ResolveCount
    , SUM(CASE WHEN CloseDate IS NULL THEN 0 ELSE 1 END) AS CloseCount
FROM dbo.Bug
WHERE OpenDate > DATEADD(dd, -30, GETDATE())
```

① 我推荐用Microsoft Excel连接数据库，将查询结果绘制为图。具体方法请参考我的博客文章：http://www.cnblogs.com/liangshi/archive/2012/04/02/2429683.html，http://www.cnblogs.com/liangshi/archive/2012/04/02/2429685.html，http://www.cnblogs.com/liangshi/archive/2012/04/02/2429688.html。

度量2：统计员工提交、解决和关闭缺陷的数目

然后，他可以使用代码清单8-4所示的SQL语句来查询提交缺陷最多的10个人，其结果如图8-3所示。该SQL语句用WITH子句对OpenBy（提交者）分组，获得每位员工在最近30天所提交的缺陷数，然后利用ORDER BY BugCount DESC和SELECT TOP 10查询出提交缺陷最多的10个人。利用该查询，测试人员可以知道谁提交了最多的缺陷，并评估自己发现的缺陷数在团队中处于什么位次。

代码清单8-4 近1个月来提交缺陷最多的10位员工

```
WITH BugHunter AS (
    SELECT OpenBy, COUNT(*) AS BugCount
    FROM dbo.Bug
    WHERE OpenDate >= DATEADD(dd, -30, GETDATE())
    GROUP BY OpenBy
)
SELECT TOP 10 OpenBy, BugCount
FROM BugHunter
ORDER BY BugCount DESC
```

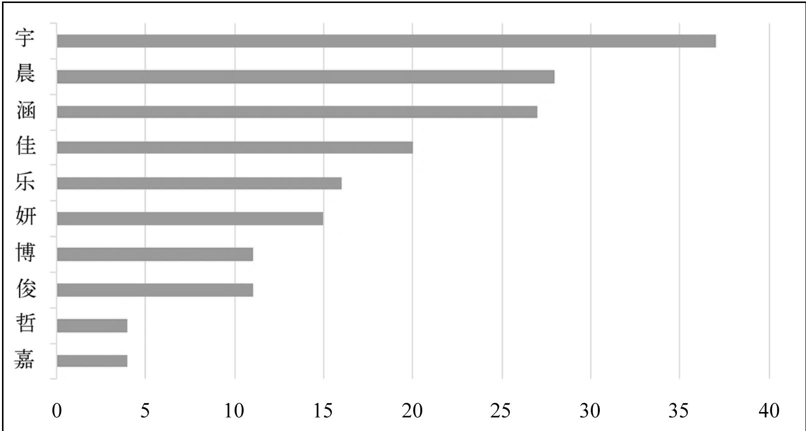


图8-3 近1个月来提交缺陷最多的10位员工

通常，那些发现较多缺陷的员工对产品比较了解，能够想出许多针对产品弱点的测试想法。测试人员应该保持谦虚的心态，认真阅读他们的缺陷报告，以积累产品知识和测试想法。在此过程中，他可以用以下问题来驱动阅读。

- 缺陷提交者是如何发现该缺陷的？他利用了哪些产品知识？利用了何种测试策略？
- 我能否利用这些知识和策略来增强我的测试？能否在此基础上发展出更具威力的策略？
- 该缺陷是什么类型的缺陷？它反应了产品在哪方面存在弱点？
- 缺陷管理系统中还存在类似的缺陷吗？能否提炼出这些缺陷的共性，从而获得失败模式？能否针对此类缺陷，设计几种有针对性的测试策略？

除了分析OpenBy（缺陷提交者），测试人员还可以采用与代码清单8-4相似的查询来分析ResolveBy（缺陷解决者）和CloseBy（缺陷关闭者——通常也是缺陷修复的检验者）。这样可以了解谁修复了最多的缺陷、谁检查了最多的修复，从而了解项目团队的工作情况。

度量3：统计每个模块的新增缺陷数

测试人员可以使用代码清单8-5分析缺陷在不同模块的分布。这段代码假定缺陷报告的路径的格式是“父模块\子模块\孙模块”，即路径是以“\”为分隔符的模块序列。代码使用“RIGHT([Path], CHARINDEX('\', REVERSE([Path])))”获得最后一个“\”之后的模块名，即路径末端的模块名。图8-4显示了查询结果，它指出模块M1的缺陷异乎寻常地多。这暗示测试人员应该去阅读该模块的缺陷报告，以调查它为什么有如此多的缺陷。在许多时候，发现缺陷较多的模块还隐藏了不少尚未发现的缺陷。通过阅读缺陷报告，测试人员可以发掘缺陷多发的根源，从而制定出有针对性的测试策略。

代码清单8-5 近1个越来缺陷在不同模块的分布

```
SELECT [PATH]
    , RIGHT([Path], CHARINDEX('\', REVERSE([Path]))) AS Module
    , COUNT(*) AS BugCount
FROM dbo.Bug
WHERE OpenDate > DATEADD(dd, -30, GETDATE())
GROUP BY [Path]
HAVING COUNT(*) > 4
ORDER BY BugCount DESC
```

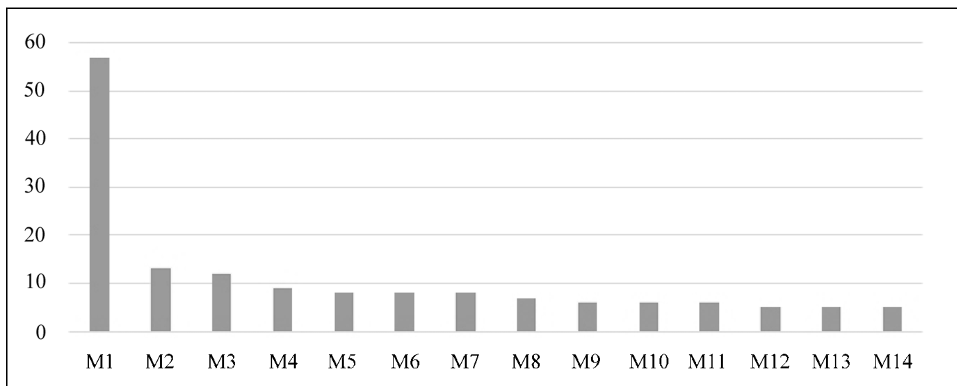


图8-4 近1个月来缺陷在不同模块的分布

度量4：统计缺陷解决方案的分布

测试人员可以分析自己所提交的缺陷是如何被解决的。代码清单8-6通过WHEREOpenBy = '涵'筛选出特定测试人员提交的缺陷，用GROUP BY Resolution对解决方案分组，进而统计出每种方案的缺陷数。图8-5是用Excel绘制的查询结果分布图。该图表明，有52%的缺陷被解决为“不能重现”，

这暗示测试手段和缺陷报告存在改进的空间。测试人员应该仔细阅读那些“不能重现”的缺陷报告，分析不能重现的原因，并采取改进措施（可参考2.2.3节介绍的方法）。

代码清单8-6　缺陷解决方案分类统计

```
SELECT Resolution, COUNT(*) AS BugCount
FROM dbo.Bug
WHERE OpenBy = '涵'
GROUP BY Resolution
```

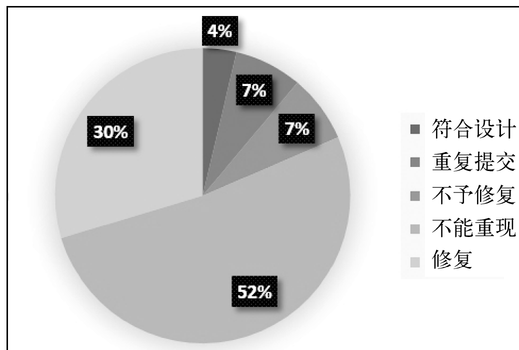


图8-5　缺陷解决方案分类统计

本节介绍了几种简单的缺陷度量方法，它们从不同的角度分析已提交的缺陷报告，可为进一步的技术调查提供线索。这些度量值只反应了产品缺陷和团队工作的一小部分信息。为了获得更透彻的理解，测试人员需要仔细阅读缺陷报告，挖掘典型缺陷、测试策略和项目风险，并将研究成果分享给团队成员。其他成员根据其知识和经验，能够补充更多的信息，从而获得更完整的成果。

8.2.2　源代码

7.1.1节已经介绍了如何通过度量源代码来获得测试灵感，本节仅做一些延伸讨论。研究源代码的基本思路仍是用简单的度量获得代码的基本信息，再通过代码阅读去产生测试想法。

代码清单8-7展示了一段PowerShell脚本，它分析当前目录及其子目录下的所有C#文件，并输出文件目录、文件名、文件行数和代码复杂度到控制台。其中，代码复杂性的度量方法是：**文件的复杂度 = 文件中分支与循环关键字的个数÷（文件的行数 + 1）**。代码清单8-7所选择的分支与循环关键字是`if`、`case`、`for`、`foreach`、`while`。这是一种基于文本分析的代码度量，能够用很简单的脚本去获得一些基本信息，为后续的代码阅读提供情报。

代码清单8-7　分析C#文件的行数和代码复杂度

```
$root = Get-Location
"Folder`tFile`tLineCount`tComplexity"
Get-ChildItem *.cs -Recurse | %{
```

```
        $folderName = $_.DirectoryName.Substring($root.Path.Length);
        $lines = Get-Content $_.FullName
        $lineCount = $lines.Length
        $opCount = ($lines | foreach { $_.split() } |
            where { $_ -in ('if', 'case', 'for', 'foreach', 'while') }).Length
        $complexity = $opCount / ($lineCount + 1)
        "$folderName`t$($_.Name)`t$lineCount`t$complexity"
    }
```

测试人员用代码清单8-7分析NUnit 2.6.2的代码，并将结果重定向到文本文件中，再用Microsoft Excel打开该文本文件，获得如图8-6所示的数据表。

	A	B	C	D
1	Folder	File	LineCount	Complexity
2	\src\NUnitCore\core	MethodHelper.cs	191	0.255208333
3	\src\NUnitCore\core	TypeHelper.cs	211	0.245283019
4	\src\NUnitCore\interfaces\Extensibilit	ITestCaseProvider.cs	60	0.213114754
5	\src\NUnitCore\core	TestCaseBuilderAttribute.cs	18	0.210526316
6	\src\NUnitFramework\framework\Cor	Numerics.cs	363	0.18956044
7	\src\NUnitCore\core\Extensibility	TestCaseProviders.cs	133	0.186567164
8	\src\NUnitCore\interfaces\Extensibilit	IExtensionPoint.cs	65	0.181818182
9	\src\NUnitCore\core\Extensibility	DataPointProviders.cs	116	0.179487179
10	\src\NUnitCore\interfaces\Extensibilit	IDataPointProvider.cs	62	0.174603175

图8-6　NUnit2.6.2的代码行数和复杂度

基于该数据表，可以快速找出复杂度最高的10个文件，结果见图8-7。复杂度高的文件往往包含复杂的逻辑，在代码修改的过程中更可能引入缺陷。因此，测试人员可以浏览这些高复杂度的文件，了解它们的功能，并构思相应的测试用例。

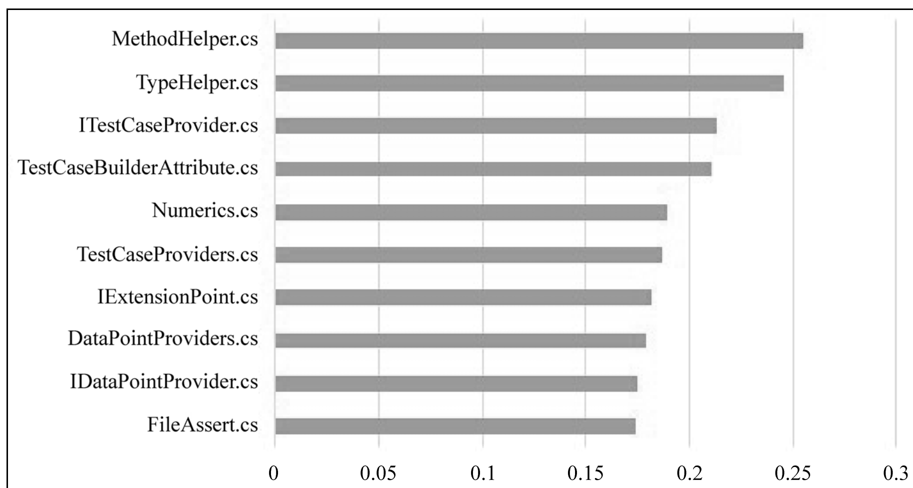


图8-7　NUnit2.6.2中复杂度最高的10个C#文件

代码清单8-8展示了复杂度最高的MethodHelper.cs的部分代码。这段代码检查输入参数(arg)的类型，并根据它的类型和值决定显示字符串（display）的值。因为要处理每种可能的类型和特殊取值，所以它拥有许多嵌套的if语句，导致了最高的复杂度。针对此类代码，测试人员可以从以下角度构思测试。

- 代码是否处理了arg的所有情况？长长的if-else语句有没有漏掉arg的一个可能类型？有没有漏掉arg的一个特殊取值？此类遗漏会导致何种问题？能不能构造测试输入来暴露问题？
- 代码处理arg的逻辑是否正确？字符串display的构造逻辑是否正确？
- 现有的测试用例是否实现了100%的分支覆盖率？如果没有，需要补充哪些测试用例？

代码清单8-8　复杂度最高的文件MethodHelper.cs（局部）

```
else if (arg is long)
{
    long l = (long)arg;
    if (l == long.MinValue)
        display = "long.MinValue";
    else if (l == long.MinValue)
        display = "long.MaxValue";
    else
        display += "L";
}
else if (arg is ulong)
{
    ulong ul = (ulong)arg;
    if (ul == ulong.MinValue)
        display = "ulong.MinValue";
    else if (ul == ulong.MinValue)
        display = "ulong.MaxValue";
    else
        display += "UL";
}
else if (arg is string)
{
    StringBuilder sb = new StringBuilder();
    sb.Append("\"");
    foreach (char c in (string)arg)
        sb.Append(EscapeControlChar(c));
    sb.Append("\"");
    display = sb.ToString();
}
else if (arg is char)
```

除了调查复杂的代码，测试人员还可以利用Excel的数据透视表统计出每个模块（目录）的

复杂度的累加值，结果见图8-8。这些模块通常拥有较多的代码和较复杂的功能，更容易发生错误，是重点测试的候选对象。测试人员可以浏览其源代码来识别测试对象，并确定测试策略。

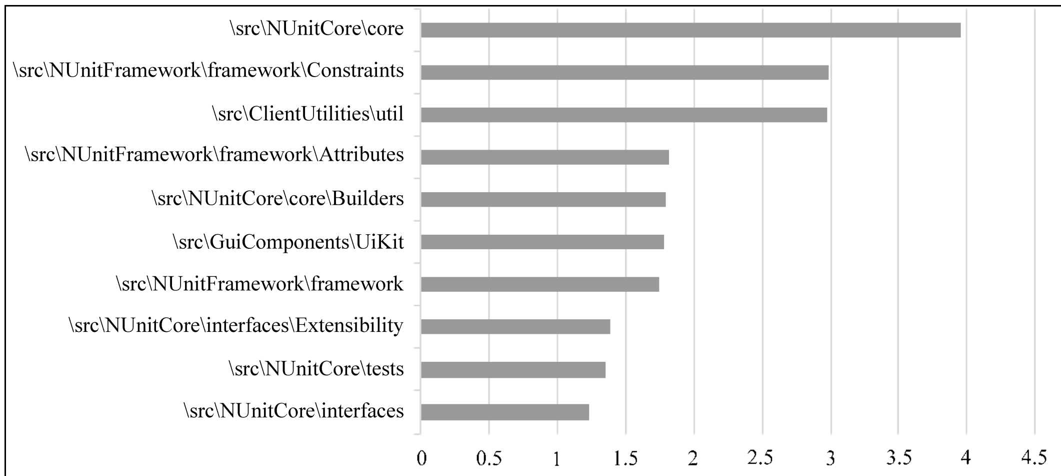


图8-8　NUnit2.6.2中复杂度累加值最高的10个目录

测试人员可以定期运行代码度量，以持续监控代码的变化。通过比较当前的度量值和之前的度量值，他可以识别出代码行数和复杂度增幅较大的文件与目录。基于该信息，他会调查代码变动的原因，并评估风险。这为及时调整测试重点提供了支持。

8.2.3　构建

大多数团队都部署了专门的构建服务器，实施持续的自动化构建。因为源代码和构建是软件研发的工作核心，测试人员应该了解代码签入和构建的规章流程，从而更好地为项目团队服务。以下是一些常见的基本问题。

- 在签入之前，产品代码需要满足什么要求？程序员需要做什么工作？测试人员需要做什么工作？
- 如果签入的代码导致构建失败，构建服务器会如何处理？谁会负责通报此问题？谁会负责修复构建？
- 构建服务器产生新构建需要多长时间？如果程序员今天下班前签入代码，明天早晨能够获得可测试的构建吗？
- 产品团队有时需要发布一个紧急的产品补丁或快速修复线上系统的缺陷。对于此类重要且紧迫的代码修改，项目团队制定了什么流程？也许，在代码签入之前，测试人员需要测试程序员提供的私有构建。对此，测试完成的标准是什么？在签入之后，测试人员需要检验新的构建确实包含预期的修复。对此，测试完成的标准是什么？

- 在签入之前，测试代码需要满足什么要求？测试人员应该如何保证自己的代码达到团队制定的质量标准？

在了解了基本流程之后，测试人员可以考虑如何利用源代码服务器和构建服务器为测试服务。以下是一些常见的做法。

- 在代码变更集被提交后，大多数源代码管理系统会发送邮件，列出签入和被修改的文件。测试人员可以在邮件系统中设置规则，为此类邮件自动加上标签，或移动到指定的邮件目录。然后，他可以每天花几分钟时间，浏览最新的代码签入邮件，识别出与自己所负责模块紧密相关的代码改动。对于这些改动，他可以再多花一些时间，查看哪些具体代码发生了改动，并评估其风险。软件项目时刻变动，测试人员很难准确预测所有的代码变更。持续监控代码签入能够帮助测试人员及时发现值得测试的内容。
- 通常构建服务器将产生的构建放在一个共享文件夹中。测试人员可以编写一个脚本去获得最新的安装文件，或指定版本的安装文件。假设，构建服务将所有构建保存在共享文件夹\\builds\product_abc中，每个构建拥有单独的子文件夹，文件夹的命名模式是“[构建号]_[构建类型]_[平台]”。其中，构建号形如1.0.1（由数字和点构成的字符串），构建类型的取值是release、debug和cover，平台的取值是x64和x86。因此，构建的文件路径类似于\\builds\product_abc\1.0.1_x64_ship、\\builds\product_abc\1.0.2_x86_debug等。代码清单8-8是一个IronPython脚本get_build.py，它搜索存放构建的共享文件夹，获得所有子目录，并反转子目录列表，使得构建号高的文件夹排在前面。之后，它扫描文件夹列表，如果某个文件夹符合用户指定的构建号、构建类型或平台，它就会输出安装文件的路径，并将该路径复制到剪贴板。如果用户不指定构建号、构建类型或平台，脚本会使用这些参数的默认值：构建号是“.”（它可匹配任意构建号），构建类型是ship，平台是x64。此时，脚本会选择最新的x64平台的ship类型的构建。利用该脚本，测试人员只需运行一条命令，就可以获得最新的或特定的安装文件的路径，而省去了手工搜索构建路径的开销。
- 基于代码清单8-9，测试人员可以编写一个脚本，以自动安装最新的构建。利用操作系统的定时任务，他可以安排测试机在每天清晨自动反安装产品，然后安装最新构建。当他上班的时候，测试机已经准备好了最新版本的产品供他测试。

代码清单8-9　get_build.py

```
import System, sys, clr
clr.AddReference("System.Windows.Forms")
import System.Windows.Forms.Clipboard as Clipboard

def find_value(argv, *values):
    for v in values:
        if v in argv:
            return v
    return values[0]
```

```
def find_build_number(argv):
    for a in argv:
        if '.' in a:
            return a
    return '.'

argv = sys.argv[1:]
build_number = find_build_number(argv)
flavor = find_value(argv, 'release', 'debug', 'cover')
platform = find_value(argv, 'x64', 'x86')

root = r'\\builds\product_abc\2.0'
folders = System.IO.Directory.GetDirectories(root)
for folder in reversed(folders):
    if (build_num in folder
            and build in folder
            and platform in folder):
        setup = r'%s\setup.exe ' % folder
        print setup
        Clipboard.SetText(setup)
        break

# usage:
# ipy.exe get_build.py [build_umber] [release|debug|cover] [x64|x86]
```

8.2.4 自动化测试

许多项目团队都使用自动化测试来检查构建的质量，典型的工作流程如下。

(1) 构建系统生成新的构建，然后通知自动化测试系统。

(2) 自动化测试系统配置测试环境，并部署新构建。

(3) 自动化测试系统从自动化测试仓库获得BVT集合。

(4) 自动化测试系统运行测试集合，并将结果存入测试结果仓库。

(5) 如果BVT结果显示新构建质量良好，可以进一步测试，自动化测试系统获得其他可用的测试集合。

(6) 自动化测试系统运行测试集合，并将结果存入测试结果仓库。

其中，被执行的自动化测试用例往往由测试小组编写，模块的测试负责人会编写该模块的测试用例。为了更好地维护测试用例集合，测试人员应该周期性地调查几个基本的问题，并根据调查结果制定相应的行动方案。

问题1：自动化测试的反馈频率如何？

运行自动化测试的目的是持续、快速、稳定地获得产品质量的反馈，所以测试人员需要了解自动化测试的反馈频率。在许多小型项目中，项目团队会每天运行所有的自动化测试用例。因此，测试人员可以确信新签入的代码在24小时之内会被所有测试用例检查。在某些大型项目中，自动

化测试的数目非常庞大，测试小组将它们分成几个集合，以不同的频率运行。对此，测试人员需要调查几个具体问题。

- 自动化测试拥有哪几个测试集合？例如，一些测试小组会根据测试的性质，将测试用例划分为：BVT集合、基本功能测试集合、完整功能测试集合、性能测试集合。
- 每个测试集合以什么频率运行？每次运行需要多长时间？例如，在某个测试小组中，BVT每天运行多次（耗时2小时），基本功能测试集合每天运行1次（耗时12小时），完整功能测试集合每周运行1次（耗时48小时），性能测试集合每2周运行1次（耗时48小时）。
- 在自己负责的测试用例中，哪些应该以更高的频率执行？哪些可以用较低的频率执行？

调查以上问题的目的是优化测试集合，使高频率运行的测试集合用较短的时间发现严重的缺陷（这些缺陷往往使后续的测试集合遭遇大规模的失败），使低频率运行的测试集合用较充裕的时间发现余下的缺陷。基本目标是既可以尽早发现严重的缺陷，又确保高频率的测试集合在规定时间内完成。测试人员应该检查自己的测试用例，将重要的用例移入高频率的测试集合，将不太重要的用例移入低频率的测试集合。对于新加入团队且接手一批自动化测试的测试人员，这是一项“必修”的功课。

问题2：自动化测试覆盖了哪些测试点？没有覆盖哪些测试点？

测试人员应该定期检查自己负责的自动化测试，以评估他们覆盖了哪些需求、功能、质量特性、用户情景等测试点。其目标是了解自动化测试已经测试了哪些内容——这些内容无需手工重复测试。更重要的是了解自动化测试没有测试哪些内容，并制订相应的行动计划。以下是一些常见的测试方案。

- 补充自动化测试用例以提高测试效率。有些功能是产品的基础，测试人员应该尽快发现其中的严重错误。然而，每天手工地检查会耗费太多资源，且占用了测试其他功能的时间。一种合理的策略是编写测试用例，覆盖重要功能的基本使用情景，并将测试用例加入每日运行的测试用例集。这样，测试人员可以在24小时内发现破坏产品基础功能的严重缺陷。
- 补充自动化测试用例以提高测试能力。有些极具价值的测试只能由计算机执行，例如6.4节介绍的大规模自动化测试，它利用计算机（集群）的强大性能执行海量测试用例，提供了手工测试无法匹敌的强力检查。此外，有些测试涉及复杂的计算，手工执行容易出错且令人疲倦。对此，可考虑构建数据驱动的自动化测试，通过持续增强测试数据，来逐渐提高对各种计算情况的测试覆盖。
- 补充自动化测试用例以弥补测试漏洞。在项目过程中，测试人员和产品试用者会发现许多缺陷。某些缺陷很容易通过自动化测试来捕获，但是由于自动化测试集合不够严密，它们成为“漏网之鱼”。在处理软件缺陷时，测试人员可以自问：我可以通过增强自动化测试来发现此类缺陷吗？需要付出多少时间才能完成？如果所需要的时间在允许范围内且没有更重要的任务，测试人员可以编码实现新的测试用例。

- ❑ 补充手工测试或半自动化测试以确保在产品发布前测试相关内容。在紧张的进度压力下，可能没有时间构造大量的自动化测试。测试人员需要制订切实可行的测试计划，综合利用自动化测试、半自动化测试和手工测试，让测试工作在项目结束前达到测试完成的标准。

问题3：有哪些测试用例需要重构？有哪些测试用例可以退役？

测试人员应该定期检查自动化测试的结果，以发现那些失败率特别高的测试用例，并分析失败的原因。许多时候，测试用例经常失败是因为测试代码不够健壮，不能够妥善处理产品运行过程中的种种情况。此类发出虚假警报的测试用例给自动化测试带来了负面影响。一方面，测试人员花费了宝贵的时间去调查测试失败，却不能发现产品缺陷；另一方面，虚假的警报降低了测试用例集的可信度，久而久之，测试人员会对测试失败习以为常，降低侦测产品缺陷的警觉性。所以，测试人员应该严肃对待不良测试代码，尽可能将其重构为稳健的代码。软件开发专家Gerard Meszaros的著作*xUnit Test Patterns: Refactoring Test Code* [Meszaros07]详细描述了测试代码的“坏味道”，并给出了一批重构手法和测试设计模式，值得测试人员参考。

随着产品的发展，有些测试用例的逻辑已经不能反映当前产品的状态，属于过时的测试代码。还有一些测试用例的检查逻辑已经被其他测试用例所实现，属于“冗余的”测试用例。为了提高测试集合的运行速度和稳定性，测试人员可以考虑让这些测试用例退役。如果测试人员担心去除测试用例会降低测试覆盖率，他可以检查代码覆盖率来确认删除测试是“安全的”。在不显著降低覆盖率的情况下，一个稳定且快速的测试用例集要明显优于一个缓慢且脆弱的测试用例集。

问题4：除了定期运行的自动化测试，还有哪些测试用例和测试工具？

在一些历史悠久的项目中，项目团队常常积累了大批的自动化测试用例。某些测试用例并没有加入正式的测试集合。相关测试负责人会在合适的时候（如产品发布前、项目里程碑结束前）运行它们，以实施特定角度的测试。例如，一些项目团队会安排专人负责压力测试，他维护了一组自动化测试。在为期一个月的迭代开发中，他会选择一个周末，让自动化测试反复执行，以暴露崩溃、死锁、内存泄漏等问题。周一，他会调查测试日志，并提交缺陷报告。新加入团队的测试人员应该了解这些被暂时“隐藏”的测试用例，考虑能否利用它们来提高自己的测试效率。例如，他可以考虑为自己的模块编写一些自动化测试，加入压力测试集合，从而让周末执行的压力测试能够发现该模块的缺陷，以及该模块和其他模块交互所导致的错误。

除了自动化测试用例，项目团队会拥有一批实用工具。它们通常切合软件产品或技术平台的特征，能够帮助测试人员高效地完成配置环境、部署产品、运行测试、调试诊断等任务。有些工具虽不常用，但对于解决特定类型的问题，是不可或缺的利器。测试小组新人应该询问领导和同事，了解团队的测试工具及其用途。这不但会提高测试人员的工作效率，而且可以帮助他更好地理解产品和技术。

由本节的讨论可知，研究与维护自动化测试的目的是提高软件测试的效率。为此，测试人员需要根据项目的测试策略，注重实效地发展自动化测试。随着项目的发展，他需要持续地分析并调整自动化测试，让它快速、稳定、有效地提供软件的质量反馈。

8.3　基于风险的测试

聚焦风险是一种基本的测试策略，而风险是大多数测试活动需要考虑的基本元素。对于产品而言，风险是产品可能遭遇的失败。无论失败原因是逻辑错误还是能力局限，只要潜在的失败危害了用户价值，就构成了风险。对于项目而言，风险是令项目产生不良结果的事件，或不良结果本身[DeMarco03]。在概念上，项目风险涵盖了产品风险，即产品可能遭遇失败是一种典型的项目风险。

基于风险的测试（简称“风险测试”）是针对特定风险设计并运行测试，以暴露导致项目失败的问题。5.1节和5.5节已经介绍了一批基于风险的测试技术，本节将继续讨论一些常用的测试方法。

8.3.1　通过测试调查风险

在软件项目中，风险是可能存在的导致项目受挫的问题。该描述有两层含义：第一，风险并不是灾难，它只是可能发生，而非不可避免；第二，风险可能转化为灾难（称为“风险暴露”），给项目带来损失。因此，测试人员在考虑测试活动的优先级时，要分析相应风险的暴露概率和损失大小。

- 风险暴露的概率：风险由潜在问题转化为具体失败的概率。概率的高低与产品和团队的特征相关。例如，C++程序出现内存泄漏的概率较高，C#程序不会出现传统意义上的内存泄漏（即不被引用的动态内存总是会被垃圾回收器回收）。又例如，一个组建多年的C++团队积累了一批好的开发实践、程序库、调试工具和测试工具，他们所开发的C++程序出现内存泄漏的概率会比较低。如果团队成员多数是C++新手，缺少足够的经验和工具，他们所构建的程序就较容易出现内存泄漏的问题。
- 风险暴露的损失：灾难失败所导致的价值损失。通常，破坏用户数据的失败会导致重大的损失。试想一下，如果软件失败导致用户的数据库被破坏，用户基于错误的数据进行了大量的销售交易，这势必造成严重的财务损失。此外，软件崩溃等错误会阻塞用户的工作流，使得他们不能正常完成业务，也会造成大量的（财务）损失。如果在项目后期甚至在产品发布后才发现这些严重的缺陷，软件项目会遭到沉重的打击，软件企业可能需要向用户支付巨额赔偿。

根据概率和损失，测试人员可以评估风险的优先级。

- 一种定量的优先级评估方法是：**风险的优先级 =风险暴露的概率×风险暴露的损失**，其中风险暴露的概率是0到1之间的浮点数，风险暴露的损失用赔偿金额来评估。该方法的优点是可以获得数值形式的优先级。这样，测试人员能够对所有风险进行线性排序，使测试重点一目了然。其不足是测试人员有时很难用金额去准确评估风险的损失，而且不同的人可能给出差异很大的估算值，令优先级的评估结果存在争议。
- 测试人员也可以定性评估风险的优先级。其中，概率的取值集合是｛很低、低、中、高、很高｝，损失的取值集合是｛很小、小、中、大、很大｝。一般而言，高概率、大损失的风险拥有最高优先级，值得重点测试。该方法的不足是不能对风险项进行线性排序，不能立即判定两个风险项的优先顺序。例如，一个风险项是高概率、小损失，另一个风险项是低概率、大损失，谁的优先级更高并没有统一的评判方法。这时，测试人员可以咨询产品经理、领域专家和用户，请他们帮忙判断。

风险测试的主要任务是分析项目风险，然后设计相应的测试来暴露失败。这并不是一个线性的过程，它是随项目进展而迭代展开的。测试专家Jams Bach认为测试本身就是风险分析，即测试人员应该通过风险测试去迭代地评估风险的概率、损失和优先级。他用图8-9解释了风险测试的循环过程[Bach11]。

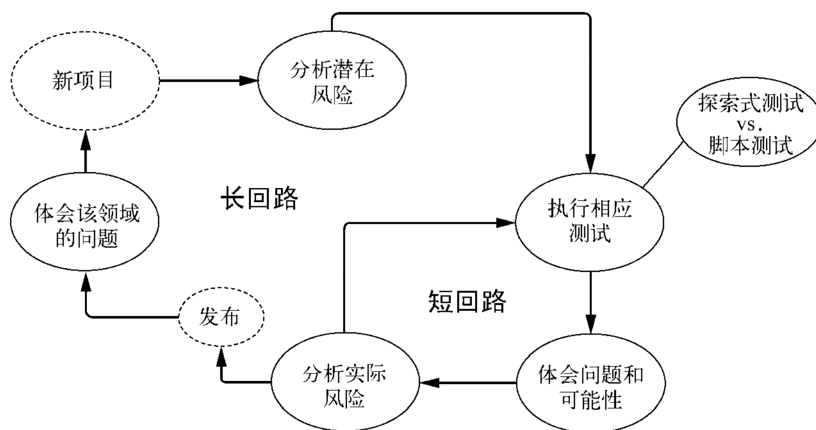


图8-9　基于风险的测试项目迭代

由图8-9可知，在测试项目中，存在风险分析的长回路和短回路。长回路贯穿整个项目周期，利用整个项目来分析领域风险。长回路包含如下阶段。

(1) 进入新项目。

(2) 分析可能的风险，拟定需要测试的风险列表。这时，测试人员对产品和领域还缺乏了解，很难提出针对性很强的风险列表。为了提高风险列表的质量，他需要运用自身的知识和经验，还要阅读相关测试文档、咨询领域专家、请教有经验的同事等。

(3) 短回路：实施具体的测试，然后根据测试反馈，设计新的测试，发掘新的风险。

(4) 回顾整个项目过程，分析所发现的问题，列出项目领域的常见风险。这为今后的项目提供了第一手的风险分析资料。

在短回路中，测试人员实施具体的测试，并根据测试所发现的新信息来评估风险。短回路包含如下活动。

(1) 根据当前的风险列表，实施相应的测试。测试人员可以利用预先编制的测试脚本，也可以实施探索式测试——根据测试结果，立即设计新的测试。通常，测试人员很难预测产品存在哪些风险、风险隐藏于哪些模块、风险会以何种形式暴露等问题，基于动态反馈的探索式测试更适合风险探测的任务。
(2) 测试人员评估已经发现的问题，根据实际情况推测其他风险存在的可能性。
(3) 测试人员根据评估结果，调整风险列表。他会补充新浮现的风险，并根据测试情况修改已有风险的概率和损失。
(4) 根据新的风险列表，测试人员设计并执行新的测试。

总体而言，James Bach的流程建议测试人员充分利用测试的迭代性，在项目全过程建立风险分析的反馈回路。这有助于尽快识别新出现的问题，并让测试逐渐逼近那些隐藏的问题。

8.3.2 失败模式

根据上一小节的讨论，风险列表是测试设计的重要参考，它记载了产品可能存在的问题。其核心是描述产品失败方式的失败模式，它刻画了失败的情景和特征 [Kaner11]。

快速测试是一种典型的基于风险的测试技术（参见5.5节）。它由一组测试想法构成，每个测试想法都针对某个风险（失败模式）。表5-4、表5-5和表5-6就列举了大量的测试想法和它们所针对的风险。这些表来源于测试专家多年的积累。他们在测试工作中收集了许多具体的缺陷，经过分析提炼，获得了一组有代表性的失败模式。针对这组模式，他们提出了一组测试想法，并让它们经受测试实践的打磨。其成果是一组易学易用的失败模式和测试想法。

测试专家Cem Kaner认为失败模式是一种通用的风险测试技术，可以有效地产生大量的测试想法。为此，他提供了一组问题，帮助测试人员发掘失败模式 [Kaner11]。

- 针对产品的每个模块或功能，测试人员思考其失败模式和测试方法。
  - 该功能可能会触发哪些失败，即包含哪些失败模式？
  - 对于特定的失败，它具有哪些外部特征？
  - 如何检测该失败？
  - 检测该失败需要多少测试资源？
- 在获得失败模式后，测试人员可以评估其影响，即执行影响分析。

- 该失败可能有哪些影响？
- 其影响包含哪些变化因素？
- 该失败的最严重损失是什么？平均损失是什么？
- 修复导致该失败的缺陷需要多少资源？
- 综合考虑失败损失和检测失败的成本，该失败模式值得被测试吗？

利用以上问题，测试人员可以获得初始的失败模式列表。除此之外，他还可以参考其他测试文档，以补充更多的失败模式和测试想法。常见的参考资料包括测试指南（3.2.7节）、测试想法列表（3.2.8节）、检查列表（3.2.11节）、缺陷目录（3.2.12节）等。综合考虑他人的研究成果可以更快地产生测试想法，并弥补自身知识和经验的不足。

随后，测试人员将失败模式列表作为风险列表，用来指导风险测试。他将一个失败模式看作一个风险，设计并执行具体的测试来发掘缺陷。在此过程中，他不但会发现新的软件缺陷，还会学习到新的业务知识、产品知识和技术知识。结合这些新发现，他应该积极评估现有失败模式的有效性，并补充新的失败模式，以产生新的测试想法。该迭代过程就是图8-10中的短回路。

经过短回路的反复锤炼，在项目结束时，测试人员可以获得一份针对当前领域和产品的失败模式列表。这为下一轮长回路的测试提供了坚实的基础。这份列表不但是测试计划的重要参考，还是很好的培训资料[Kaner11]。通过阅读它，测试小组的新成员能够了解产品特征，实施快速测试。通过测试，他们可以更好地学习产品，并初步体会当前领域的主要风险。

8.3.3 项目级别的风险

除了软件缺陷，测试人员还需要考虑项目级别的风险。此类风险如果转化为灾难，有可能导致项目失败。表8-1是Cem Kaner总结的项目级别的风险[Kaner11]。

表8-1 Cem Kaner总结的项目级别的风险

风险	描述
新事物	新代码尚未经过严格测试，可能包含较多缺陷
新技术	新技术可能引入一些难以预料的问题
学习曲线	在学习新事物、新技术时，团队成员更容易犯错
变动的事物	代码变更可能使已有代码不能正常工作
糟糕的控制	如果团队不使用源代码管理系统，代码文件会被篡改或意外删除
迟来的变化	有时，团队必须在发布前紧急改动代码。紧张的进度和匆忙的工作很容易导致错误
强压的工作	有时，团队承担了太多的工作，以致于很难高质量地完成它们
疲劳	疲惫的团队会犯更多的错误
分布式的团队	分散在不同地点的团队会面临交流困难，缺乏交流会导致不一致的理解，而不一致的理解会导致不能协同工作的代码

（续）

风　险	描　述
其他人员问题	团队由形形色色的人构成，他们可能遭遇各种问题，如酗酒、离异、亲人去世、两个程序员彼此不说话等。这些都会对代码产生负面影响
惊讶的功能	某些功能未得到仔细地设计，以致于对其他功能产生了不良影响
第三方代码	团队可能不太了解外部代码，也很难修复其中的缺陷
未列入计划	一些计划外的、突然出现的任务有可能被仓促完成，包含许多错误
模糊	文档中的模糊描述会导致不正确或不一致
冲突的需求	彼此冲突的需求有可能损害一部分关系人的利益
神秘的沉默	当一些重要的议题没有被讨论或记录，它们可能没有得到充分地考虑，或者它们的设计者试图隐藏其中的问题
不知晓的需求	团队可能漏掉了一些正当的需求，以致于损害了产品的质量
衍变的需求	在整个项目周期，需求都在大幅度变化，这很容易导致项目失败
缺陷丛生	产品的每个部分都存在许多缺陷
最近的失败	如果一个功能最近暴露出许多缺陷，那么它可能还隐藏了不少其他问题
上游依赖	一个模块依赖于它的上游模块。上游模块的改动或延期会导致该模块出现问题
下游依赖	一个模块依赖于它的下游模块。该模块的改动或延期会造成下游模块出现问题
发散	在时间或空间上发散又必须协同工作的一组事物，它们的复杂性会引入缺陷
无确定的主题	任何没有明确主题的函数或数据都容易被误用
复杂	难以理解的事物很难被正确地构建
语言相关的错误	编程语言会引入一些风险，例如C语言的悬挂指针、内存泄漏等
极少的单元测试	缺乏单元测试的项目不具备基本的“开发安全网”
狭窄的测试策略	狭窄的测试策略会遗漏大量的缺陷
无力的测试工具	如果测试工具不能帮助测试人员发现和分离某类缺陷，那么该类缺陷可能会被漏测
无可修复	一些缺陷报告被解决为“不予修复”，因为没有人知道该如何修复它们
不可测试	如果测试人员只能缓慢、困难、低效地测试某个功能，那么该功能很可能没得到充分地测试
负面报道	如果某个特性的失败会导致大量的恶评和负面报道，它就值得重点考虑
法律问题	如果某个特性的失败会导致诉讼，它就值得重点考虑
高危	如果某个特性的失败会导致重大伤害，它就值得重点考虑
精密	如果某个特性需要其行为或结果与需求严丝合缝，它就值得重点考虑
容易滥用	如果某个特性需要小心使用或用户培训，它就值得重点考虑
受欢迎的	如果某个特性特别受欢迎、被广大用户使用，它就值得重点考虑
战略的	如果某个特性对业务特别重要，它就值得重点考虑
重要人物	如果某个特性被重要人物所使用，它就值得重点考虑
显著性	如果某个特性的失败会导致显著的后果或令用户愤怒，它就值得重点考虑
隐蔽性	如果某个失败会被产品暂时隐藏，却最终导致灾难，它就值得重点考虑

测试人员可以将表8-1当作通用的风险列表。在项目之初，他根据该列表研究项目和团队，分析出值得持续监控的风险，并将相应的测试想法写入测试计划。在项目过程中，他需要持续监控产品的风险，并适时地调整风险列表和测试策略。

对于许多项目级别的风险，单个测试人员并没有资源加以完整地监控。但是，测试人员有责任在自己负责的领域实施风险测试，以积极探测项目级别的风险。一旦发现某个风险有可能转化为灾难，他应该立即向测试经理汇报。当项目管理者及时地获得警报时，他们就可以采取必要的行动去避免灾难。

8.4 小结

本章讨论了如何从测试的角度来研究项目和团队，介绍了一些实践方法。

- 测试人员应该主动了解项目团队的使命、目标和运作方式。
- 研究项目团队的最佳方法是执行具体的测试任务，并积极地咨询与反思。
- 使用语境独立的启发式问题可以多角度地研究项目语境。
- 人，在一起工作的人，是项目语境中最重要的部分。
- 为了更好地与同事协作，测试人员需要了解他们的思考风格。
- 积极主动的讨论是良好协作的基础。
- 从软件缺陷、源代码、构建、自动化测试等项目元素中，能够获得许多有益的测试信息。
- 利用简单的度量就可以获得许多有价值的情报。
- 基于风险的测试针对项目风险设计并运行测试，以发掘可能存在的问题。
- 测试本身就是风险分析，它是一个迭代过程。
- 在测试实践中积累失败模式可以帮助测试小组更好地测试。
- 测试人员有责任监控和报告项目级别的风险。

第 9 章

团队工作

大多数测试人员都工作在一个团队中，其工作的主要内容是持续地向项目团队和项目关系人提供测试服务。有时他会长时间地独立测试，但最终他还是需要与团队成员协作，使技术调查的结果推动产品的发展。在很大程度上，测试人员的工作价值取决于他对项目团队的贡献。

本章探讨测试人员如何有效地在团队中工作，以及如何恰当地实施测试管理。本书的目标读者是第一线的测试人员，所以本章并不讨论测试经理如何管理测试小组，而是分析测试人员如何面对项目管理的一些挑战。

9.1　工作风格

不同的项目团队拥有不同的工作风格，这要求测试人员根据团队情况来调整自己的工作方式。不过，在职业生涯中，测试人员应该秉持一些基本态度，让它们指导具体的工作。本节分享我的一些心得，仅供读者思考。

9.1.1　测试人员通过服务团队来体现自己的价值

在项目层面，测试人员应该持续地提供高质量的测试服务，帮助项目团队成功地交付产品；在企业层面，测试人员应该帮助公司去达成业务目标。这是对“专业人员”的基本要求，但并不容易做到。

首先，测试人员应该设定正确的工作目标。管理学大师Peter Drucker指出，执行经理是掌管自己时间的价值并影响机构执行能力的人，而大多数知识工人都是执行经理[Drucker93]。在实际测试中，测试人员可以决定技术调查的范围、深度和手段。即便项目团队给出了详细的测试脚本，测试人员在测试运行、问题调查（蕴含设计新的测试用例）、缺陷提交、状态报告等具体任务中，仍具有很大的自主性。因此，专业的测试人员会主动将自己看作执行经理，去积极地设定工作目标，有效地利用时间。

为了设定正确的目标，他需要思考一系列问题。

- 公司的业务战略是什么？
- 当前项目的使命是什么？它如何支持公司的战略？
- 项目团队的具体目标是什么？它们如何实现项目团队的使命？
- 我有哪些具体目标？如何让自己的目标支持团队的使命？
- 做哪些具体的工作才能实现自己的目标？

回答这些问题并不容易，但只有经过慎重的思考，才能让个人目标符合团队利益。许多时候，如果测试人员不掌握足够的信息，就难以制订合理的工作计划。这时，他可以请教测试经理和项目负责人，请他们解释团队目标和优先级。好的领导会为团队指明方向，并帮助测试人员安排具体工作项的优先级。

在确定具体的工作项时，测试人员应该放宽视野，不必计较工作划分的边界。因为软件测试是通过技术调查来提供产品的质量信息，所以测试人员的工作核心是为项目团队和关系人提供信息服务（参见1.2节）。因此，当团队或关系人需要质量信息时，只要有足够的空闲时间，测试人员就应该主动提供服务。此外，测试人员应该意识到他是测试小组的一份子，当测试伙伴需要帮助时，他也应该积极响应。多承担一些工作会让测试人员更忙碌，但这也是积累经验、提升技能、提升影响力的好办法。总体而言，一个积极的测试人员能够更快地成长，更有可能脱颖而出。

在设定目标之后，测试人员需要认真地完成自己的工作。坦率地说，许多必不可少的测试工作是比较枯燥的。例如，项目团队需要发布一个紧急的产品补丁。在修复代码签入之前，测试人员测试该代码变更，检查它确实修复了问题且没有引入新的缺陷。在代码签入之后，构建服务器产生了发布给用户的补丁。为了发现签入和构建过程的错误，测试人员需要测试发布版本的补丁，主要工作包括：安装产品、测试补丁的安装、再次检查补丁确实修复了问题、测试补丁的反安装。在大多数情况下，对发布版本的测试不会发现新问题，而且这部分工作也没有技术挑战，因为签入前的测试已经解决了所有技术问题。但是，该工作是必须的，因为项目团队必须防备一些难以预料的小概率事件，例如程序员在签入时意外地修改了一行代码、另一个程序员同时签入的代码意外地覆盖了本次修复、另一个团队同时签入的代码与本次修复不兼容等。所以，专业的测试人员会理解任务的意义，用负责而不是敷衍的态度去处理一些“无趣”的任务，并让工作质量达到团队认可的标准。

另一种积极面对“无聊”工作的方法是用创新思维来重新思考任务。例如，在测试补丁的活动中，机械且冗长的手工操作包括：安装产品、安装已经发布的补丁、安装被测补丁等。为了提高效率，测试人员可以编写一个脚本，去自动完成这些安装操作。这将一个枯燥的手工任务转变为一个自动化挑战。也许编写自动化脚本的时间会比手工安装的时间多5倍。但是，通过编写脚本，测试人员提升了编程能力，学习了安装产品和补丁的知识，这对于他未来的工作大有裨益。此外，该脚本可以分享给测试小组，以提升团队的整体效率。可见，用进取心看待平凡的工作，就能够发现它不平凡的一面。

9.1.2　测试人员应该正直

对测试人员而言，正直是客观公正地处理工作并始终诚实地报告质量问题。在许多项目环境中，该价值观会受到各种挑战，测试人员需要妥善应对。

对于一些程序员而言，缺陷报告传递的是一种“负面”信息。当程序员接到缺陷报告后，他不得不放下更有趣的新功能开发，转而去调查某个疑难杂症。这打断了他的工作流，引入了任务切换的开销，带来了额外的工作。一旦决定修复该缺陷，他还需要编写、测试、签入代码，这会耗费他更多的时间。此外，有些程序员认为缺陷报告是对其工作的批评与质疑，这在无形中令他对缺陷报告怀有戒备之心。因此，专业的测试人员应客观地传递信息，应该报告的是产品需要解决的问题，而不是对任何人的批评。用责备或轻视的语气报告缺陷对于解决缺陷并没有帮助，反而激起了充满自我保护的辩论，让相关人员的注意力偏离了缺陷本身。

对于一些项目负责人而言，缺陷报告也不是“好消息”，它们常常代表了严重的产品问题，增大了项目延期的可能性。因此，在项目结束前夕，他对缺陷报告的第一反应是：“为什么（你）现在才发现这个问题？”或“这为什么是一个问题？”。如果其语气暗示出质疑或责备，那么传递“坏消息”的测试人员将承担很大的压力。面对此类情况，测试人员应该坚持如实报告问题，因为作为信息服务者，诚实可信是测试人员的最基本要求。隐瞒或扭曲信息会伤害测试人员的信誉，也会伤害项目。在这里，我分享一个自己的教训[①]。



在一个Web服务发布前夕，我发现了一个严重的崩溃缺陷。在接到通知后，程序员提供了一份包含修复的私有构建。经过近一周的压力测试，我相信当前的私有构建不存在崩溃问题。我将这一信息口头报告给程序员，通知他可以签入修正代码。

在发现崩溃、诊断故障、测试私有构建、报告测试结果的过程中，我并没有在缺陷管理系统中记录相应的缺陷。事后证明我没有用缺陷来追踪崩溃问题是严重失职，但那时我也有“充分”的理由。

首先，当时处于发布前夕，缺陷修复规则是任何新发现的缺陷都要在24小时之内修正，或放到下一次发布再修正。这次的崩溃非常复杂，我分析内存转储文件时没有把握可以在24小时内找到解决方法。即便拿到了私有构建，我也需要几天的时间来做压力测试，才能检验修复的有效性。我不记录缺陷，是为了争取更多的时间。

其次，管理层对24小时的时间窗口非常重视，总是在催促程序员修正错误，或将不严重的错误推迟到未来的发布。这在心理上给程序员巨大的压力。大家在潜意识中认为，如果不能在24小时内“搞定”缺陷，自己的工作绩效会受到影响。程序崩溃是严重的问

---

①关于该缺陷的来龙去脉请参考我的博客文章：http://www.cnblogs.com/liangshi/archive/2010/09/05/1818554.html，http://www.cnblogs.com/liangshi/archive/2010/09/07/1820540.html， http://www.cnblogs.com/liangshi/archive/2010/09/09/1822015.html。

题，但是其根源是一个很难重现的.NET CLR（Common Language Runtime）缺陷。提交这样一个可能不那么“要紧”的缺陷，如果程序员不能在24小时修正，会不会对他的个人绩效产生负面影响？他会不会受到管理层的质疑？

当时，我以为程序员在得到修复有效的测试结果后，会在某次签入中“顺便”包含该代码变更。但是，在发布的最后阶段，只有修正缺陷的代码变更才能签入。那位程序员手上恰恰没有缺陷（这是我始料未及的），他也觉得CLR的缺陷在产品环境中恐怕不会复现，于是没有签入修正。

大约一个月以后，该服务在产品环境中崩溃。从症状上看，就是我所发现的缺陷。我到那时才知道，产品代码中不包含程序员的修正。现在看来，不正式记录该缺陷有以下问题。

第一，管理层不知道该缺陷的情况。他们将根据不准确的信息作出发布决策。

第二，在发布前，所有的流程都围绕缺陷展开。不记录缺陷，就不能“合法”地签入代码，就不能修复崩溃。

第三，不记录缺陷，就没有一个文档正式记录问题症状、调试结果、修复进度、测试进度。缺陷记录可以协调程序员、测试人员、项目管理者来共同解决严重的问题。

第四，缺陷是团队学习的最佳素材之一。不把如此有教益的缺陷记录下来，是团队积累的一个损失。

幸运的是，服务重启后能够继续工作，对业务影响不大。经过讨论，我们决定在两周后发布补丁，以修正该崩溃和其他一些问题。

再次反思该缺陷处理过程，不难获得如下结论。

第一，流程的出发点可能和结果背道而驰。要求在24小时内修复新问题，是为了让代码中的缺陷持续减少。但是，许多缺陷难以在24小时内被修复。在压力下，项目团队可能选择不在当前发布修复该缺陷。这使得代码中的缺陷被“隐藏”了，而不是被修复了。

第二，管理层应该营造一种鼓励报告缺陷的氛围，这使得发布决策能够基于真实的情况。

第三，当缺陷处理（报告、修复、检验）与个人绩效联系在一起时，缺陷报告与缺陷修复的积极性可能会被压抑。

第四，即便流程不会评估个人绩效，只要开发者心理上感觉它们之间有联系，他们的行为也会受到影响。

第五，作为专业的测试人员，发现问题一定要报告。报告可以讲策略和时机，但是不能将问题“吞没”。对于这次的崩溃问题，我应该第一天就提交到缺陷管理系统。由于它复现的概率不高，我们可以将它推迟到下一个发布周期。与此同时，我们可以尽可能寻找解决方案。一旦找到修复方法，就将该缺陷移回到当前发布周期，然后签入代码将其修正。

Tom DeMarco和Tim Lister在经典著作《人件》中指出软件项目的主要问题是社会学问题

[DeMarco99]。因为测试人员的主要工作是传递对项目和团队有影响的信息，他会面临更多的人际关系方面的挑战。对此，他应该秉持正直的价值观，以诚实的态度提供信息服务。这比随波逐流、人云亦云要困难许多，但却是发展职业生涯的正路。

9.1.3　测试人员的影响力来自于出色的工作

如2.1节所论述，测试人员报告缺陷是为了让缺陷得到修复，这往往要求他说服程序员和缺陷评审小组去采取修复行动。一个总是让缺陷得到修复的测试人员，无论他的头衔是什么，总是拥有实质性的影响力。这种影响力的根源来自于测试人员的优秀工作，以及在工作中表现出的技术能力。

对于测试人员而言，出色的工作通常体现为：他的测试策略是产品相关、聚焦风险、多种多样、注重实效的（参见5.7节）；他让测试学习、测试设计、测试执行和测试评估成为相互支持的活动，通过快速地测试迭代，持续发现产品缺陷；他对于发现的缺陷进行必要的技术调查，为缺陷的评估、复现、修复等活动提供高质量的信息；他撰写了客观、严谨的缺陷报告，让程序员和缺陷评审小组能够快速地理解缺陷的来龙去脉；他积极主动，乐于助人，用高效的技术调查来回答团队成员的疑问。一个工作出色的测试人员会逐渐在团队中建立自己的信誉，那些见证其工作、接受其服务的团队成员自然会认可他的表现，并相信他可以提出有价值的观点。

除了测试技术，测试人员还需要扩展自己的知识面。在与产品经理讨论产品时，他需要掌握足够的领域知识和产品知识。在与程序员讨论设计时，他需要掌握足够的开发知识。一个测试人员常面临的困境是，当他提出一些技术建议，程序员可能流露出不屑的态度："你作为一个（不了解软件设计的）测试人员，有什么资格要求程序员采纳你的意见？"从长远看，克服该困境的唯一途径是测试人员具备接近程序员的软件开发能力。这要求测试人员学习产品平台、程序语言、设计模式和调试技术，并通过测试实践逐步理解产品架构和代码。具备产品知识和开发技术的测试人员能够提出有洞察力的意见，虽然并不是所有的意见都会被接纳，但是好的意见可以为他赢得项目团队的认可，并逐渐增强他的影响力。

9.1.4　信任程序员的努力，并用技术调查检验其工作

如果测试人员希望团队成员可以信任自己，他就需要先信任别人。一个内心"疑邻盗斧"的人会流露出对他人的不信任，这反过来也会导致他不被信任，因为团队成员不认为他是客观公正的。在绝大多数团队中，测试人员应该秉持的基本理念是：所有团队成员都希望项目获得成功，都在尽自己的努力来提高产品的质量。那么，测试人员应该无条件地相信程序员对代码改动的描述吗？答案是，测试人员需要用技术调查来检验其描述的正确性。

本书开篇就指出软件的复杂度已经超越人的理解能力。面对复杂且变化的代码，任何人都不能保证他的代码变更总是正确无误。信任一个人，并不意味着相信他不会犯错。任何人都有盲点，

都有思考不周的情况，即便努力工作也会在代码中引入一些错误。测试的任务就是用全新的眼光考察产品，发现其中的问题。当程序员说“这段代码改动影响很小，没有问题“时，他的言下之意其实是“在我所知道的情景中，这段代码改动没有问题”。在测试时，测试人员应该参考程序员的见解，安排一些测试去检查程序员已知的情景，更要通过各种途径去挖掘程序员遗漏的情景。那些“意料之外”的情景很可能会暴露软件缺陷。

9.2 测试管理

在测试管理上，测试经理从项目整体出发，会组织与安排各种测试活动。作为执行经理，测试人员负责具体的测试任务，也应该积极计划并管理自己的工作。本节讨论一些测试管理的基本关注点。

9.2.1 个人测试计划应该是项目测试计划的延伸

通常，测试经理或资深测试人员会编写项目的测试计划，将整个项目过程划分为几个里程碑，并定义每个阶段的起止时间、预定目标、测试任务和测试执行者（参见2.1节）。作为具体任务的负责人，测试人员需要在总体测试计划的基础上，将测试任务转化为一系列小的测试目标，并拟定相应的测试策略。这是一个具体化的过程，将一个概括描述的任务转化为一组明确的工作项。对于每个工作项，测试人员都可以思考以下几点。

- 测试对象：该工作项测试哪些模块、功能、质量特性和用户情景？“测试完成”的定义是什么？
- 测试策略：使用哪些方法来测试？有哪些测试想法？需要针对哪些项目风险？
- 测试资源：完成任务需要哪些软件和硬件资源？是否需要编写测试工具？是否需要其他测试人员的协助？
- 测试时限：测试活动何时开始和结束？在这期间，投入多少精力在该测试任务中？如果测试活动不能按时完成（可能原因包括测试资源不足、阻碍测试进展的严重缺陷、突然出现的高优先级任务等），应该如何处理？
- 测试成果：测试活动的成果是什么？需要提交哪些测试报告？需要将哪些自动化测试交付使用？需要交付哪些测试工具？

经过细化，测试人员可以将一个里程碑级别的任务分解为一组工作项。在安排工作项的进度时，他应该形成稳定的测试节奏，即利用测试的迭代过程，以固定的周期来执行和报告工作项。以下是几种常见的测试周期方案，表9-1是它们的示例。

- 以里程碑为测试迭代周期。许多研发团队采用迭代开发的流程，迭代周期通常为2~4周。测试人员可以沿用此开发周期来安排测试工作的迭代。因为测试人员需要快速、持续地反馈产品质量，所以测试迭代周期在实践上应该小于里程碑。如果里程碑的持续时间为

2~3个月，那么测试迭代周期不宜超过4周。

- 以周为测试迭代周期。测试人员以周为时间单位规划自己的工作，以执行工作项和发送测试报告。该方法利用自然周这个天然的工作单元，能够方便地规划工作，并持续地报告产品的质量信息。
- 以构建为测试迭代周期。对于一些需要在短时间内完成的测试任务（如测试需要快速发布的产品补丁），测试人员可以用构建来安排测试工作。通常的策略是，第一轮测试全面覆盖代码变更，第二轮测试针对第一轮测试发现的缺陷，且检查缺陷修复是否引入新的缺陷，第三轮测试针对第一、二轮发现的缺陷，并执行回归测试。

表9-1　以测试周期安排工作项

<table>
<tr><th>周</th><th>里程碑（项目计划）</th><th>以里程碑为测试迭代周期</th><th>以周为测试迭代周期</th><th>以构建为测试迭代周期</th></tr>
<tr><td>1</td><td rowspan="2">1</td><td rowspan="2">工作项A
工作项B
工作项C</td><td>工作项A</td><td>工作项A：构建1
工作项A：构建2
工作项A：构建3</td></tr>
<tr><td>2</td><td>工作项B
工作项C</td><td>……</td></tr>
<tr><td>3</td><td rowspan="2">2</td><td rowspan="2">工作项D
工作项E
工作项F
工作项G</td><td>工作项D
工作项E</td><td>……</td></tr>
<tr><td>4</td><td>工作项F
工作项G</td><td>……</td></tr>
</table>

对于不同的工作项，测试人员可以采用不同的工作周期。对于紧急的工作，他需要以构建为周期。对于不紧急且低优先级的工作，他可以采用里程碑为周期。对于大多数工作，以周为迭代周期是较好的选择。如果一个工作项较大，不能在一周的时间内完成，测试人员可以将其拆分成若干小工作项，以放入周迭代中。

9.2.2　制订个人测试计划时应该综合考虑各种项目元素

3.2.1节介绍了启发式测试计划的语境模型HTPCM，它根据项目语境（软件开发、测试团队、测试实验室、项目需求），来决定测试任务和测试过程。这是一个有价值的参考模型，提出了一组影响测试任务和过程的项目元素。测试人员可以用它评估当前状态，从而制定具体的测试工作项。此外，他还可以利用启发式测试策略模型（参见4.2.1节）和项目风险列表（参见8.3.1节）来多角度地思考项目风险，并安排相应的测试活动。在此基础上，本节将从测试协作的角度来讨论个人测试计划。

许多时候，测试人员直接与程序员协作。程序员发布构建给测试人员，测试人员测试构建并报告缺陷。随后，程序员修复缺陷，发布新构建，测试人员开始新一轮的测试。在此过程中，测试人员需要考虑如下问题。

- ❑ 整个代码编写与测试的循环需要在何时结束？最终的代码需要在何时签入？
- ❑ 为了确保代码达到预期的质量标准，应该做哪些测试？完成这些测试需要多长时间？
- ❑ 如果程序员迟迟不能发布构建，使得可用的测试时间少于必须的测试时间，应该如何处理？
- ❑ 如果代码存在许多严重缺陷，使得测试工作被阻塞，以致于难以按时完成，应该如何处理？
- ❑ 能否与程序员沟通，请他提前发布（私有）构建，使得一些测试工作可以提前开始？
- ❑ 如果测试构建需要大量的时间，能不能向测试经理申请更多的资源？例如，请测试经理安排几个同事一起测试该构建，通过并行工作来加速测试。

为了缓解代码不能按时完成的风险，测试人员需要加强与同事的合作。一方面，他要与程序员密切交流，了解他的工作进度，并尽可能提供快速的测试服务。许多程序员愿意提前发布私有构建，让测试人员了解新设计。虽然这些构建可能不提供完整的功能，但是提供了设计和实现的详细信息，是很有帮助的测试参考。作为回报，测试人员向程序员报告私有构建中的问题，让他可以快速地修复缺陷。对于私有构建，测试人员和程序员可以约定最方便的缺陷报告方法，例如口头报告、邮件报告、正式报告等。良好的互动可以让代码更快地稳定下来，从而按时签入。在代码签入后，测试人员需要遵循日常流程，用正式的缺陷报告记录构建中的问题。

另一方面，测试人员需要及时向测试经理报告测试进度。如果测试存在延后的风险，或已经延后，测试人员应该立即向测试经理报告。不要隐藏此类“坏消息”，这会让测试人员处于不利的局面。测试人员的工作是提供产品的质量信息，而产品不能按时完成就是很重要的质量信息，需要让项目关系人知晓。隐藏不报是一种失职行为，会让测试人员的信誉受损。在报告之后，测试经理可以与其他项目管理者协作，运用更多的资源，以帮助测试人员和程序员来解决问题。例如，他们会指派资深程序员来解决复杂的缺陷，或安排更多的测试人员来加速测试。保持信息畅通更有助于问题的解决。

除了与程序员协作，测试人员之间也有许多合作机会。例如，在大型项目的集成测试中，测试小组的所有成员都会去测试同一个系统，甚至多个测试小组需要相互配合来完成整个系统的测试。在此过程中，测试人员需要考虑如何进行测试协作。以下是一些启发式问题。

- ❑ 掌握集成测试的时限和目标。
  - 集成测试预期在何时完成？总体目标是什么？
  - 一共有几轮测试？每一轮测试预期何时完成？每一轮测试的目标是什么？
- ❑ 了解测试合作者和他们的职责。
  - 有哪些测试小组参与集成测试？各个小组负责完成什么任务？
  - 有哪些测试人员参与集成测试？每个人负责完成什么任务？
  - 我的测试小组负责什么任务？
  - 我的测试小组有哪些人参与集成测试？每个人负责完成什么任务？

- ❑ 理清测试依赖，并为可能出现的问题制定应对方案。
  - ■ 还有哪些组件会影响我所测试的组件？这些组件的测试负责人是谁？我们如何协作？
  - ■ 我所测试的组件会影响哪些组件？这些组件的测试负责人是谁？我们如何协作？
  - ■ 如果某个组件存在严重缺陷，使得我无法测试我的组件，我应该如何处理？
  - ■ 如果我的组件存在严重缺陷，使得其他测试人员无法测试其组件，我应该如何处理？
- ❑ 考虑测试人员之间的协作方式。
  - ■ 通常，集成测试总负责人会召开例会，邀请所有参与集成测试的测试人员（或测试小组的代表）参与。例会将通报每个组件的测试进度，并协调测试工作。如果没有这样的例会，谁可以召集测试人员来协调工作？我可以召集吗？
  - ■ 例会有哪些固定的议题？我所关心的议题会被讨论吗？我可以提出新的议题吗？
  - ■ 例会如何处理集成测试中的协作问题？如何形成决议？
  - ■ 除了例会，还有哪些正式或非正式的测试协作方式？
- ❑ 定期报告测试进度。
  - ■ 谁会发送测试报告给所有关系人，以报告整体的测试进展？我如何向他提供信息？
  - ■ 我如何发送测试报告给我的项目团队，以便从我们的视角来检视测试进展？
  - ■ 我的测试经理期望我如何报告测试进展和测试结果？

集成测试是一项风险性比较高的测试活动，在测试环境、测试情景、上下游依赖、缺陷侦测、故障诊断、缺陷修复、进度同步等方面较组件级测试复杂得多。为了及时处理测试过程中浮现的问题，测试人员需要与测试同事紧密协作。定期召开的例会是常见的协作活动，能够协调不同部门的测试人员，以共同解决一个部门不能处理的问题。如果当前的集成测试没有例会，测试人员应该在测试经理许可的情况下，主动组织测试协调会议，并邀请相关测试负责人参与。一般而言，例会应该聚焦大部分测试人员关注的议题：介绍当前集成测试的进度、公布那些影响或阻碍集成测试的问题、讨论多部门协作才能解决的问题、安排下一轮测试的工作等。在会议上，测试人员应该积极发言，提出急需解决的缺陷和重大项目风险，争取其他部门测试人员的支持，也向他们提供帮助。除了例会，测试人员之间还可以通过面谈、电话、邮件、即时通信等多种方式快速地交换信息，以提高测试协作的效率。

在集成测试过程中，测试人员应该频繁地向测试经理报告测试进展。因为集成测试可能涉及多个部门，测试人员未必了解其他部门的情况。拥有更多信息的测试经理能够帮助他更好地理解测试伙伴和系统全貌。此外，测试经理拥有更多的测试资源和交流渠道。一些重大问题需要他和其他测试经理一起协作解决。及时地报告问题将帮助测试管理者了解产品风险，并采取必要的行动，以扭转危局。

从测试管理的角度，测试人员需要与工作伙伴通力协作，并密切追踪工作项的实际进展。测

试工作位于整个研发流程的下游，且紧密依赖于上游活动的产出。在许多项目中，测试进度延后是测试人员常常遇到的情况，是测试管理必须考虑的风险。在制订个人测试计划时，测试人员需要考虑这些风险，并安排相应的任务去监控风险和化解问题。

9.2.3　测试需要动态管理

在许多软件项目中，测试活动具有高变动性，往往以预料之外的方式展开。以下是一些常见的原因。

- **在测试之初，测试人员很难预测产品会具有哪类缺陷，他所拟定的测试策略可能大获成功，也可能无法奏效**。随着测试的进展，他会持续发现新信息，并学到许多产品和业务知识。基于新情况和新知识，他应该调整原有的测试计划，放弃无效的测试想法，补充新的测试想法去针对真正的风险。这是探索的过程，作为探险者的测试人员掌握一份地图（原有的测试计划），但需要根据地形随时修正前进路线。
- **测试工作处于整个研发流程的下游，会依赖一些测试人员不可控制的上游因素**。例如，测试人员为测试某个功能安排了一周的时间，然而该功能的交付延迟了一周，但是产品发布日期没有变动。这意味着测试人员损失了一周的时间，面临更紧张的进度压力。又例如，产品暴露出一个严重的缺陷，使得某些用户情景无法测试。程序员迟迟不能修复该缺陷，这导致测试无法推进，测试时间被无形地压缩。面对这些情况，测试人员必须调整测试计划，重新安排测试工作项和测试迭代。
- **一些测试工作对测试环境和资源有较高的要求，会依赖一些项目团队不可控制的因素**。例如，项目团队与伙伴团队做系统集成，程序员可以根据约定好的接口编写代码，而无需考虑伙伴团队的具体实现，但是测试人员必须在集成测试中覆盖两个系统的协作情景。如果伙伴团队不能及时地提供可测试的系统，集成测试计划就被打乱。又例如，程序员在单元测试中只需要使用少量的业务数据，但是测试人员在系统测试中必须使用真实的（或仿真的）大批业务数据。这些测试数据很可能需要其他团队提供。如果数据不能及时到位，那么系统测试计划就需要调整。
- **开发过程中会有一些突发事件，使得测试人员不得不中断当前工作去紧急处理**。例如，测试实验室遭遇停电事故，有一些计算机在恢复供电之后不能启动。兼任实验室管理员的测试人员不得不调查计算机故障，尽力修复问题，使得其他测试人员可以使用计算机。又例如，在线系统发现了一个安全漏洞，需要紧急修复。程序员加班加点，发布了一个补丁，接下来测试人员实施高强度的补丁测试。这样的修复与测试过程可能会持续几天，使得原计划的工作项被延迟。

可见，测试过程发生变动是“正常”情况，是复杂的技术问题和紧迫的开发过程在测试领域的自然结果。测试人员应该主动地面对变化，用动态管理来积极调整测试计划。

第一，在考虑特定测试对象时，测试人员应该为研究与调查预留足够的时间。在规划测试时，测试人员可能还没获得产品构建。此时，他要尽可能发散性思考，借助测试指南、功能列表、检查列表、测试想法列表、缺陷目录等资料（参见3.2节），提出一组差异化的测试想法，全面覆盖当时能想到的测试类型。根据已知的测试想法，测试人员提出工作量估算，并附加一些缓冲时间。在实际测试时，测试人员要积极评估测试想法的有效性。一旦发现某个测试想法没有帮助，他应该立即放弃。这样做不是为了减少测试工作量，而是为有潜力的想法提供更多的测试时间。随着测试的深入，测试人员会产生新的测试想法。他需要通过测试来检查这些想法的有效性，并将时间投放在那些有效的想法上。

第二，测试人员需要动态地调整工作项和测试想法的优先级。“要事第一”是软件项目管理的基本原则。如果项目团队总是先做高价值的功能，当他们必须舍弃没有完成的工作而立即发布时，被舍弃的是相对不重要的功能。这使得产品可以提供尽可能多的价值。测试工作也是如此，测试人员应该先执行重要的任务，使得重要的缺陷尽早被发现。为此，他需要周期性地评估产品状态和项目风险，调整测试工作项和测试想法的优先级。如果他对一些任务的优先级有疑问，可以咨询测试经理，请他提供信息。如果他对某个测试想法的有效性没有把握，可以安排一个短的时间盒，用实际测试去检查想法。基于优先级安排工作，测试人员能够将大部分测试时间投放在重要的领域，仅用少量时间去快速检验不重要的领域。

第三，测试人员可以借助测试迭代，定期评估项目情况，并调整测试工作项。在大多数项目中，以周为测试迭代周期能够获得较好的效果。在里程碑之初，测试人员实施“里程碑计划”，确定里程碑需要完成的任务，制定工作项列表，并把它们安排到合适的星期中。里程碑计划能够揭示一些项目风险。例如，计划结果是每周都有许多工作，这暗示测试任务没有缓冲的余地，一旦某个任务超时，很可能没有时间完成余下的任务。另一个典型的风险是，一项重要的测试工作被安排在最后一周。如果程序员不能按时交付构建，测试人员无法在里程碑结束前完成测试。对于这些风险，测试人员需要与测试经理、程序员和相关人员协商，共同制定缓解方案。

在获得里程碑计划之后，测试人员以周为单位，迭代地实施测试，其过程如图9-1所示。在周一，他评估当前的测试情况，更新本周的工作项列表，加入新出现的任务，删除不再需要的任务，并为每一项任务制定优先级。通过“周计划”，他获得了一份符合项目实情的工作计划，为实现 “要事第一”提供了基础。经过一周的工作，他在周五发送测试报告给测试经理和项目管理者，综述测试情况，并报告新浮现的项目风险。一方面，测试报告帮助经理了解项目进展，为他们的决策提供必要的信息。另一方面，撰写测试报告的过程也是评估测试进度的过程，能够帮助测试人员更好地实施下周的计划。以周为单元，迭代地执行“计划→实施→总结”能够帮助测试人员动态调整测试计划，以应对持续变化的软件项目。

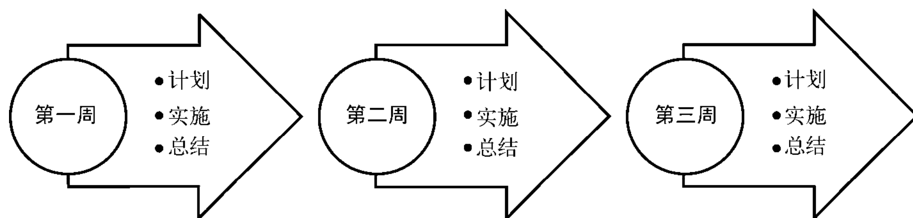


图9-1 以周计划驱动的测试迭代

第四，测试人员应该关注项目风险，并及时向程序员、产品经理和测试经理报告。测试专家Cem Kaner等指出，测试人员是项目的前灯（the headlights of the project），应该为项目决策提供信息基础[Kaner01]。恰如本章反复强调的，测试人员很可能没有足够资源去解决他所面临的问题。隐藏严重的缺陷、落后的进度很可能导致项目延期、产品失败等严重后果。及时地报告阻碍测试的问题、测试进度的延后、产品所暴露的缺陷、项目所遇到的风险，是测试人员不可推卸的职责。这些信息将“照亮”项目前方的道路，让项目管理者看清实际路况。因为他们掌握更多的资源和信息，且可以作出重大发布决策（包括延迟发布、删减功能集等），所以更可能扭转项目的前进方向，以避免在黑夜中坠入悬崖。

退一步说，报告测试工作的困难或延迟是一种“自我保护”的手段。面对严重落后的进度时，有些测试人员尝试隐瞒“坏消息”，并期望加班工作可以挽回进度。这样的策略有时可以奏效，更多的时候只能让坏消息在最后一刻暴露出来。那时产品面临发布或里程碑结束，项目团队已经没有时间来采取补救措施，测试人员将面对他无法承担的巨大责任。为了避免这样的情况，测试人员应该尽早报告他所遭遇的困难，以获得测试经理和团队成员的帮助。这保护了他的职业发展，也帮助了项目团队。

第五，测试人员需要经常自问：“我现在可以测试什么？能够如何测试？”[Kaner01]。这是两个看似平常，但是具有提示意义的启发式问题。在忙碌的项目中，测试人员可能无暇思考，只是遵循已有的测试计划。这样的忙碌状态会让他错过一些有价值的测试情报，漏掉一些改进测试策略的机会。在每个里程碑和每个迭代周期，都用两个问题去检视软件项目和测试策略，可以帮助测试人员实施动态管理，让测试策略随着项目一起发展。

在某种程度上，本节的内容是用动态管理来应对项目级别的风险（参见8.3.3节）。也就是说，测试人员需要将项目风险作为驱动测试计划发展的重要因素，持续地监控风险，并制定相应的行动方案。毕竟好的测试策略是产品相关、关注风险、多种多样、讲求实用的[Kaner01]。

9.3 软件估算

实施有效计划的一个重要前提是准确地估算工作项所需的时间或成本。程序员需要估算编写和单元测试一个功能需要多长时间，测试人员需要估算完成该功能的所有测试需要多长时间，这

些估算值构成了项目整体计划的基础。软件行业的普遍情况是，项目团队会低估（甚至故意低估）完成工作所需要的时间。在许多项目中，过低的估算和不良的管理导致进度紧张、发布延期代价高昂。

软件开发专家Steve McConnell的名著《软件估算——“黑匣子”揭秘》（*Software Estimation: Demystifying the Black Art*）详细讨论了软件估算的核心概念和一些估算方法，具有很高的参考价值[McConnell06]。本节将介绍该书的一些关键观点。

9.3.1　测试人员应该估算自己的任务

在软件开发中，以下几个拥有不同含义的词汇常常被无意识地混用，导致了误解和歧义。因此，有效估算的第一步是正确理解基本概念。

- 估算是对任务所需的时间或成本的预测。估算应该是客观的分析过程，旨在获得准确的预测值，而不考虑最终结果是否超出了可用的时间和资源。
- 目标是期望完成的任务。在软件项目中，常见的目标包括编码完成、测试完成、发布等。
- 计划是实现目标的方案。与估算不同，计划是主观的目标求解过程，旨在充分利用项目时间和资源，以最大的概率去达成目标。
- 承诺是许诺在特定日期之前完成目标。在软件项目中，承诺意味着“使命必达”的责任，以及不能兑现诺言的问责。

在许多时候，项目关系人并不掌握以上概念的差别，所以他们常常将估算等同于计划或承诺。例如，项目主管问测试人员：“你评估完成集成测试需要多长时间？产品还有一个月就要上线了，时间够用吗？”在字面上，项目主管在询问集成测试的估算，其实他是在（用委婉的口气）要求测试人员承诺在一个月时间内完成测试。又例如，测试经理问测试人员：“你认为完成集成测试需要多长时间？产品还有一个月就要上线了，时间够用吗？”对于大多数技术领导而言，该问题是要求测试人员给出一个切实可行的计划，以按时完成集成测试。如果测试人员随口答道 “一个月时间足够了”，那么测试经理会认为他已经拥有了一个很好的计划，并且承诺在一个月之内完成任务。

为了避免误解，测试人员需要理解关系人的真实意图，并就所讨论的内容达成共识。以下是几个需要交流的要点。

- **估算、计划和承诺的核心是目标**。只有清楚地定义目标，才能进行有意义的估算、计划和承诺。因此，测试人员需要向关系人确认任务的内容，以及关系人对完成质量的期望。如果任务是集成测试，那么测试人员需要询问测试的范围和测试完成的标准。
- **测试人员需要向关系人询问，他究竟需要什么信息，以及为什么需要这些信息**。如果关系人索取估算，他有可能在进行项目级别的估算，希望测试人员提供局部的详细信息。

如果关系人需要计划，他有可能期望了解测试活动的关键时间点，以协调多项测试活动。如果关系人期待承诺，他有可能期望测试人员确认测试任务可以按时完成。了解关系人的动机可以帮助测试人员提供正确的答案。

- ❑ **测试人员应该与关系人协商，确定一个提交正式答案的时间**。除非测试人员对目标进行过认真的估算和计划，否则他不应该当场给出估算、计划或承诺。在交谈的时候，测试人员仅能凭借经验进行估算，很难得出准确的估算值或稳健的方案，而根据不良估算所做的承诺很可能无法兑现。他应该向关系人说明，他需要认真分析之后，才能给出确切的答案。然后，他和关系人约定一个时间，届时他会提供正式的答案。

虽然目标、估算、计划和承诺拥有不同的语义，但是在个人测试管理中，它们通常是紧密关联的。测试人员获得一个测试任务之后，应该调查它的目标和完成标准。之后，将其分解为一组测试工作项，并估算每个工作项所需的时间，以获得该任务的时间估算。如果该任务所需的时间超过预算，测试人员会调整工作项，以减少任务所需时间。如果测试人员发现满足完成标准所需要的最短时间仍旧超出预算，他需要向测试经理说明该任务无法在期望时间内完成。可见，估算是测试计划的重要环节，是评估测试计划是否合理的重要指标，也是测试人员是否做出承诺的基础。

测试人员应该负责自己任务的估算，而不是无条件地接受他人的估算。根据别人的估算做出承诺，很可能让自己处于不利的情景。例如，测试经理让测试人员在5天内完成一项测试任务。某些测试人员会不假思索地答应下来。不幸的是，在实际测试过程中，他发现该测试任务很复杂，需要更多的时间。为了不违背自己的承诺，他硬着头皮做下去，尝试着用加班来完成任务。然而埋头工作并没有加快进度，在第5天，他不得不面对任务无法完成的局面。这时，他有两个选择：好的选择是报告测试经理，测试任务没有完成，他需要更多的时间；差的选择是隐瞒测试任务的真实进度，报告测试任务完成。无论如何，他都没有兑现“5天内按照标准完成任务”的承诺，为项目引入了风险。倘若风险暴露为事故，那么他不得不面对问责。

在上例的情景中，有经验的测试人员会采取不同的做法。在接到任务后，他会估算完成它需要的时间。如果发现该任务需要8天才能完成，他会立即与测试经理讨论该估算值。讨论的主要目的不是说服测试经理接受自己的估算值，而是理解为什么两个估算值存在差异。是两人对任务内容的理解存在差异吗？是测试经理拥有一些高效的测试方法吗？是测试经理知晓其他测试人员会覆盖部分测试内容吗？是存在某些强制条件，使得该任务必须在5天内完成吗？在理清这些问题后，测试人员和测试经理通常可以对任务的内容和完成方式达成一致意见，并获得一个双方都认可的估算值。一些可能的讨论结果包括：测试经理缩减了测试任务的范围，使得测试任务可以在5天内完成；测试经理同意8天是合理的估算值；测试经理为该任务投入更多的人手，使得估算值降到5天。

总之，估算是测试管理和项目协作的重要活动，是计划和承诺的基础。负责任的测试人员会谨慎地评估工作项的时间，并用估算值来分析工作计划的可行性。如果他人对估算值存在不同意

见，他会将分歧视为交流的机会，通过交换意见来达成共识。

9.3.2　用计数和计算作为估算手段

Steve McConnell用一个非常精彩的故事解释了估算的基本手段。本节将其引用如下。



你和几个估算人员参与一个大型的招待会，大房间中坐满了人。突然，司仪用麦克风发问："谁可以准确地估算出这个房间中有多少与会者？"

估算人员Bill说："我经常估算人数，凭我的经验，房间里大约有335人。"

估算人员Karl说："房间里的桌子排成了矩阵，共有11行7列。会议组织者告诉我一张桌子坐5个人，目测也是如此。所以，11乘7再乘5，一共有385人。"

估算人员Lucy说："我注意到该房间有容量标记，它最多可以容纳485人。看上去，目前已经坐满了70%~80%，也就是说有340~388人。取平均数就是365人。"

你说："我去调查一下，片刻就回。"几分钟后，你公布了调查结果："进入房间之前要扫描门票。我去问了门口的验票员，她说根据扫描器，一共扫描了407张票，而且目前没有人离开。所以，与会者应该为407人。"

司仪公布的答案恰恰是407人。

这个例子生动地展示了几种常用的估算方法，并说明了它们的优先顺序。

- **计数**是最优先的估算方法。只要可以用清点的方式来获得答案，就应该采用这种直截了当的方法。它通常能给出最准确的答案。
- **计算**是次优的估算方法。如果无法用计数获得答案，就应该先对某些对象进行计数，然后根据特定的算法来计算估算值。Karl先清点了桌子的数目，然后根据每桌5人的信息算出了估算值。Lucy先了解了房间的容量，然后根据感觉（房间坐满了70%~80%）算出了估算值。
- **判断**是最后的手段。Bill纯粹根据感觉提供了度量值，很大程度上是在碰运气。不幸的是，这次他的运气并不好，其度量值最不准确。

测试人员不应该将估算、计划和承诺建立在运气之上，所以他应该尽可能使用计数，然后才考虑计算，迫不得已才选择判断。可惜，在许多软件项目中，大部分估算都来自"拍脑袋"的判断。在此环境中，如果测试人员不立即给出估算值，而是说"请让我数一数（或算一算）"，他会被认为不够"勇敢果断"。对此，测试人员应该顶住仓促决定的压力，坚持用更准确的方法来提供估算结果。毕竟，最后是测试人员来承担估算有误的责任。

对于许多测试任务，测试人员很难直接使用计数的方法。对此，他可以将测试任务分解为多个工作项，然后对每个工作项进行计数。最后，他将工作项的估算值累加起来，以获得测试任务的估算值。这是一种"分而治之"的策略，将一个大型任务的估算转化为对一组小型任务的估算，

通过提高局部的准确性来保证最终结果的准确性。此外，累加一组估算值有助于误差补偿。当某些工作项的估算值偏高，另一些工作项的估算值偏低，其累加值“抵消”了正负误差，可能更加准确。

9.3.3 历史数据是估算的重要参考

为了较准确地估算，测试人员需要寻找与任务工作量相关的计数对象。如果存在与计数对象相关的历史数据，测试人员就可以方便地将计数值转化为估算值。表9-2记录了一些常见的估算对象和它们的历史数据。

根据历史数据，测试人员可以较方便地估算出一项任务所需的时间。以自动化测试结果调查为例，假设自动化测试集合包含2000个用例，测试失败率大约是2%，调查一个失败的测试用例平均是15分钟，那么一轮自动化测试调查需要$2000 \times 0.02 \times 15 = 600$分钟（10小时）。如果测试小组安排4个人共同调查，平均每个人需要花费2.5小时。

表9-2 将计数值转化为估算值的历史数据

估算对象	把计数值转化为估算值的历史数据
功能测试	• 一个功能或特性的平均测试时间。 • 每千行代码的平均缺陷数。 • 缺陷被解决为修复的概率。 • 对一个缺陷修复进行测试的平均用时。 • 一个缺陷修复引入新缺陷的概率
性能测试	• 一轮性能测试需要考察的情景的数目。 • 录制和调试一个情景的平均用时。 • 回放一个情景以收集性能数据的平均用时。 • 汇总数据、编写性能测试报告的平均用时
数据库测试	• 测试一个存储过程的平均用时。 • 测试一个函数的平均用时。 • 测试一个视图的平均用时。 • 测试一个表的平均用时
网页测试	• 测试一个报表页面（以数据呈现为主要情景）的平均用时。 • 测试一个表单页面（以数据提交为主要情景）的平均用时。 • 测试一个只读页面（只包含静态文字、图片、超链接）的平均用时
回归测试	• 一轮回归测试的平均用时。 • 一轮回归测试所提交的平均缺陷个数。 • 缺陷被解决为修复的概率。 • 对一个缺陷修复进行测试的平均用时。 • 一个缺陷修复引入新缺陷的概率

（续）

估算对象	把计数值转化为估算值的历史数据
运行自动化测试	• 测试实验室运行自动化测试集合的平均用时。 • 自动化测试集合包含多少个测试用例。 • 测试用例失败的概率。 • 调查一个失败的测试用例的平均用时
编写自动化测试	• 对于一个被测函数，平均编写的测试用例个数。 • 编写一个测试用例的平均用时。 • 调试一个测试用例的平均用时。 • 对测试代码进行代码审查的平均用时
测试环境搭建	• 配置一台测试计算机的平均用时。 • 安装被测产品的平均用时。 • 配置被测产品的平均用时

一般而言，测试人员应该尽量自动地获得历史数据。许多团队用数据库存储缺陷报告、自动化测试的信息（包含用例编号、用例名、用例所属的测试集合等）、自动化测试的结果，这为数据分析提供了便利。测试人员编写简单的SQL查询，就可以很方便地获得历史数据。2.3.5节建议测试人员在编写缺陷报告时，要支持未来的缺陷查询。这条建议在本节体现为，缺陷报告应该支持未来的数据分析。以回归测试为例，测试小组可以约定回归测试的缺陷报告的标题应该以“[RF]”（Regression Failure的缩写）为前缀。这样，代码清单9-1所示的SQL查询（有关数据表dbo.Bug的定义请参考8.2.1节）能够返回一轮回归测试所发现的缺陷个数。这段代码假设一轮回归测试只针对一个固定的构建号，它筛选出Title（标题）包含“[RF]”的缺陷，并用OpenBuild（提交缺陷的构建号）对它们分组，从而获得每个构建号所拥有的回归缺陷的数目。利用该信息，测试人员很容易计算出一轮回归测试所提交的平均缺陷个数。

代码清单9-1　统计每一轮回归测试发现的缺陷

```
SELECT
    OpenBuild, COUNT(*) AS OpenCount
FROM dbo.Bug
WHERE OpenDate > DATEADD(dd, -30, GETDATE())
    AND CHARINDEX('[RF]', Title) = 1
GROUP BY OpenBuild
```

因为部分项目数据并没有被自动地记录，所以测试人员有时要做一些数据采集。他可以参考3.2.13节介绍的基于测程的测试管理（SBTM），用测程来组织具体的测试活动。标准的SBTM建议测试人员记录测试活动的时间分配，即在产品安装、测试设计与执行、缺陷调查和报告、非测试活动中各花费了多长时间。积累这些数据将为今后的估算提供有力的支持。

此外，好的测程拥有明确的主题，这意味着它通常针对一组具体的测试对象。在一个测程结束后，测试人员只需进行简单的回顾，数一数测试所覆盖的对象，就可以获得有价值的数据。例如，他分配了一个1.5小时的测程，专门用于自动化测试的失败调查。在测程结束后，他统计一

共调查了多少个失败用例，然后计算出失败用例调查的平均用时。又例如，他分配了一个2小时的测程来测试Web页面。在测程结束后，他统计一共测试了多少个报表页面、表单页面和只读页面，从而计算出测试每种页面的平均时间。

Steven McConnell建议开发人员从四类数据开始收集：规模（代码行或软件中可计数的对象）、工作量（工作时间）、时间（日历时间）、缺陷。该建议也适用于测试人员，他可以从很少一组数据开始积累，用少量的时间去聚焦基础性的信息。在此基础上，他应该利用测试迭代来周期性地收集最新的数据。这不但能让他更好地估算，还能让他掌握项目的发展动向。

9.3.4 同时估算最差情况和最好情况

在估算时，一个很有帮助的技巧是同时分析最差情况和最好情况的估算值。如果只给出一个估算值（称为“单点估算”），测试人员常常倾向于给出一个乐观值。为了避免过于乐观的估算，测试人员应该强迫自己思考最坏情况，评估出完成任务的最长时间。这样做主要有以下3个好处。

- 当思考最坏情况时，测试人员会更发散地思考。他会挖掘出一些被乐观思考忽略的因素，想出一些不常见、但可能发生的不利情况。在某种程度上，悲观思考帮助测试人员更好地理解任务和语境，甚至让他更准确地评估最好情况。
- 估算只是一种预测，其结果不可能做到百分百准确。此外，工作能否按期完成，不仅仅取决于恰当的估算，还依赖于有力的执行、无干扰的工作和关系人的支持，其中一些因素非测试人员可以控制。单点估算不能自然地反映出项目发展的不确定性，给人一种“到那时一定可以完成”的幻觉。如果估算结果是一个时间区间，那它能直观地表达出软件项目无法忽视的高度变动性。
- 即便测试人员需要提供单点估算，他也可以先估算最好情况和最差情况。这让他更全面地分析项目环境，从而在这两点之间选择一个合理的值。在统计上，任务以最好情况或最差情况完成的概率是很低的，最终的估算值应该拥有更高的完成概率。更详细的讨论请参考Steve McConnell和Tom DeMarco等专家的著作[McConnell06][DeMarco03]。

另一个提高估算能力的方法是经常比较任务的估算时间和实际完成时间。基本策略是建立基于实际值的反馈机制，以逐渐提高估算的准确度。为此，测试人员可以在每个测试周期结束时，研究那些估算值与实际值相差较大的案例，调查其中的原因，从而提出相应的估算改进方案。利用快速的反馈，他可以更快地提高其估算能力。

9.4 度量

在日常工作中，测试人员会主动用一些定量的方法来分析软件和项目（例如7.1.1节、8.2.1节、8.2.2节、8.2.4节讨论了一些度量方法）以获得测试想法，也会被要求提供一些数值以反应项目进展。为了更好地完成这些任务，本节讨论一些软件度量的基本原则和方法。

9.4.1　理解度量方法的基本元素

有效运用一种度量方法的前提是透彻地理解它。Cem Kaner教授提出了一个评估度量方法的框架[Kaner00]，其核心元素如下。

- 意图：为什么项目团队或测试人员需要使用这种度量方法？其度量结果有何用途？
- 属性：度量方法总是针对项目元素的某个特性，反应被调查对象在这个方面的表现。
- 算法：定量评估属性的方法。
- 属性与算法的联系：一组理论或假设，以说明使用该算法来评估属性是合理的。联系通常决定了算法的适用范围、优点和缺点。
- 可能的副作用：误用度量方法带来的不利影响。许多时候，团队成员会不自觉地“优化”度量结果，使其满足管理层的要求。这样做可能会导致一些负面结果。

在实施特定的度量方法时，测试人员需要仔细考虑它的意图、属性、算法、联系和副作用。在8.2.2节，代码清单8-7展示了一种评估C#代码复杂性的度量方法，其定义是：**文件的复杂度 = 文件中分支与循环关键字的个数 ÷（文件的行数 + 1）**。表9-3总结了该方法的基本元素。

表9-3　文件复杂度的基本元素

意图	测试人员尝试发现复杂度最高的一批代码文件，将它们作为候选阅读对象。其背后的动机如下。 • 程序员在编写或修改复杂代码时，更容易出错。因此，复杂代码包含更多的缺陷。 • 复杂代码不容易测试，需要重点研究。 • 了解复杂代码所涉及的功能，能够更好地理解产品并评估风险
属性	度量方法旨在定性地评估产品代码（调查对象）的复杂性（属性）
算法	文件的复杂度 = 文件中分支与循环关键字的个数 ÷（文件的行数 + 1）
联系	该算法的本质是统计分支与循环关键字在一行代码中出现的概率。其背后的假设如下。 • 分支和循环可能带来复杂的执行路径，复杂的执行路径会导致理解困难。 • 分支和循环越密集（出现概率越高），代码就越复杂
副作用	作为代码阅读的辅助工具，该度量方法并不造成严重的项目风险，毕竟决定代码阅读质量的是测试人员的技术能力和投入时间。不过，测试人员仍旧需要知晓该方法的不足：代码复杂性是由多个因素决定的，该算法仅考虑了复杂性的一个方面，并不能全面地评估复杂性。 • 在分支与循环关键字个数相同的情况下，深度嵌套的分支与循环通常更加复杂。但是，该算法不分析嵌套深度，不能为深度嵌套的代码提供更高的复杂度。 • 在C#语言中，一些LINQ操作符提供了类似分支的筛选能力和类似循环的迭代能力。但是，该算法没有统计LINQ操作符，不能完整地提供此类代码的复杂度。 • 在C#语言中，有多种根据程序状态选择执行路径的方法，例如虚函数、反射、动态代码（通常由dynamic关键字引入）等。但是，该算法没有考虑这些情况。 • 一般认为异常处理代码比较复杂，较容易出错。但是，该算法没有考虑异常处理

结合度量的基本元素，可以获得运用度量方法的基本注意点。以下几个小节将逐一论述这些要点。

9.4.2 明确度量的目标

度量的目的是为项目决策提供信息，是为了更合理的行动。因此，测试人员需要了解度量的用途，知道它所服务的目标。然而，在项目压力下，测试人员有时会忘记度量背后的目标，只是机械性地提供度量值，或让自己的工作符合预设的度量值。这样做无助于提高产品质量，且丧失了一些改进开发过程的机会。下面是一个真实的案例。



在某个项目中，代码覆盖率是一条产品发布标准，它要求测试的语句块覆盖率达到70%。在发布前夕，测试人员Kyle正在为测试覆盖率“达标”而努力奋斗。那是一个比较大的C++程序，其中有一些代码，利用端到端的系统测试很难覆盖。于是，Kyle创造性地使用调试工具Windbg来提高覆盖率。他将Windbg附加到被测程序上，然后利用调试器指令，手工改变程序的执行路径，以快速覆盖任何他希望覆盖的代码。不幸的是，有一块代码在入口处就崩溃了。虽然该异常被Windbg捕获，但是程序无法继续执行。在试了几次之后，他把程序员叫了过来，问道：“程序在这里崩溃了，如何绕过去继续执行？”程序员大骇：“你难道不研究一下它为什么崩溃么？”Kyle怒斥之：“别说这些没用的，干正事要紧！”

这个故事是当事程序员亲口说给我听的。写在这里不是指责Kyle对程序崩溃不闻不问。恰恰相反，我认为Kyle的举措具有相当的合理性。第一，利用Windbg修改执行路径，已经扰乱了程序状态。即便程序崩溃，也很难认定这是一个缺陷。第二，崩溃发生在一段很难被执行到的代码中。即便是程序缺陷，也不会引入显著的风险。第三，项目临近结束，还有许多高优先级的测试工作。结束手上的工作，将精力集中在那些暴露真实问题的“正事”上，才是注重实效的策略。

为了正确地使用代码覆盖率度量，或者说为了让投入在代码覆盖率上的时间获得丰厚的回报，测试人员必须理解评估代码覆盖率的目的。正如测试是对代码的反馈，代码覆盖率是对测试的反馈。获取代码覆盖率，不是要证明测试的充分性，而是要发现测试的遗漏，从而补充测试用例，以发现隐藏的缺陷。所以，代码覆盖率是一种寻找缺陷的技术，是提高测试的手段，不是测试的目的。只收集代码覆盖率，却不利用它去发现更多的缺陷，是误入歧途。

然而，在项目结束前，测试人员往往有许多缺陷要做回归测试，要分析压力和性能测试的结果，要与其他团队配合进行跨系统的集成测试。在这种情况下，他们没有充足的时间来阅读代码、调查代码覆盖情况。此时的代码覆盖率统计只是一项必须完成的任务，难以给项目带来明显的收益。那么何时才是分析代码覆盖率的最佳时机呢？以下是我听到故事后的思考。



我们应该在项目之初就分析代码覆盖率，并一直持续到项目结束。测试人员应该从项目早期就开始理解代码，并逐步增加测试用例。随着项目的展开，他需要知晓代码的

变化和测试的不足，这时代码覆盖率能提供有帮助的信息。为此，他需要持续统计代码覆盖率，并用它来改进测试。以下是三种可能的做法。

第一，每日构建生成产品的发布版、调试版和覆盖版，使得自动化测试可以提供最新的覆盖率结果。通常覆盖版都是对调试版进行代码插桩所得，行为与调试版一致。除了一些性能敏感的测试用例，其他测试用例都可以在覆盖版上执行，以获得大部分测试用例的覆盖率。在我的部门，有一个项目组就是这么做的。他们的每日测试会生成一份当天的代码覆盖率报告供测试人员参考。

第二，定期生成被测试程序的覆盖版，分析代码覆盖率。现在代码覆盖率工具很多，测试人员很容易收集代码覆盖率信息。工具的门槛很低，关键在于投入精力，坚持收集并分析。

第三，不借助代码覆盖率工具，通过阅读代码、在调试器中执行测试用例来分析代码覆盖率。这种方法的优点是启动成本低，可以随时执行；缺点是难以生成美观的代码覆盖率报告。从提高测试质量的角度看，其效果与其他策略相当，甚至可能更好。因为代码覆盖率的关键不是工具，而是对代码的理解。

由以上案例不难看出，测试人员需要探究度量方法背后的目标。唯有如此，他才能在恰当的时机，以恰当的方式来实施度量，并实现目标。忽视度量背后的动机，将错过改进测试流程的机会，难以真正提高开发过程的质量。

在理解目标之后，测试人员可以更好地实施度量，让度量成为软件开发的助手。下面是一个真实的案例。



有一次，我接到一个测试任务，去测试一个Web服务。这个Web服务用C#实现，它接受外部调用，根据输入值更新后台数据库的数据。这是一个小项目，核心逻辑大约在千行上下。我接手的时候，代码已经写完。开发者有简单的测试，他提供了5条端到端自动化测试用例供我参考。

当时，我最头痛的是不知道该Web服务的业务逻辑和实现逻辑。在仔细地阅读了规格说明和几封很长的讨论邮件之后，大致知晓了项目背景，但是仍旧不了解实现逻辑，也不知道什么行为是可以接受的、什么行为是错误的。由于开发者与我有16个小时的时差，不能随时和他联系，很多问题得不到及时的解答，测试很难展开。

为了有所突破，我开始阅读实现代码。我的方法是，在编程工具Visual Studio中单步执行开发者的测试用例，观察程序中变量值的变化和数据库中数据的改变。有了初步的认知，我开始增加测试用例。大部分测试用例都是一个模式。

步骤1：恢复被测Web服务的数据库到初始状态。

步骤2：调用Web服务。Web服务是无状态的，但是它会改变数据库中的数据。

步骤3：对数据库中的数据进行一致性检查。我花了许多时间编写并增强检查代码，

整个测试开发过程是渐进的。

步骤4：执行当前测试所特有的检查。

设计完一个测试用例，我就在Visual Studio中单步执行它，从而更深入地了解实现逻辑。在阅读、执行代码的过程中，我发现了一些潜在的问题。这时，我会增加一个测试用例去“证明”当前问题可以导致产品的失败。随着测试用例越来越多，我对代码的理解也越深，发现的缺陷也越多。后来，缺陷不那么好找了，我就仔细地读代码，特别是那些没有被测试覆盖过的代码。然后，便增加测试用例去尽可能地提高测试用例的覆盖率。当时的策略是，测试用例集要尽可能地覆盖所有的代码块，它还要覆盖主要的数据状态和我觉得有风险的数据变更。

在项目结束的时候，我对测试的代码覆盖率很有自信。只有三个在catch语句中的代码块没有被覆盖，它们在产品环境中被执行的概率非常低。而且我仔细阅读过代码，它们即便被执行也不会导致失败。实际上，我比较担心的是测试没有覆盖一些复杂的数据转换。虽然我在不停地补充此类测试用例，但是这个项目的测试周期较短，测试用例能够覆盖的程序状态空间还是偏少。庆幸的是，产品上线后运行良好，没有发现问题。

回顾这个项目，我觉得有两点经验可以吸取。

第一，人的大脑才是最好的代码覆盖率工具。代码覆盖率工具能提供大量的信息。只有在充分理解代码与测试的前提下，这些信息才能得到有效的利用。这个项目的代码规模较少，因此我没有使用代码覆盖率工具。通过反复阅读代码、调试代码，我收集了代码覆盖率信息，并利用这些信息改进了测试。可以说，代码覆盖率是随着对代码不断的探索而自然提高的。对于大型项目，合理的利用代码覆盖率工具自然是大有裨益的。在这一过程中，阅读、理解、探索代码仍旧是指导测试的核心活动。

第二，代码覆盖率只提供了程序结构被覆盖的信息，测试人员还需要关注程序状态是否被有效地覆盖。在本项目中，程序的状态可以视为数据库中的数据值。Web服务能否完成很多时候取决于初始数据值是否有效、目标数据值是否可以接受。相同的执行路径对于不同的数据会产生完全不同的结果。如果仅仅关注程序结构，很可能会漏掉一些由程序状态引发的问题。

9.4.3 掌握属性和算法的联系

在常用软件度量中，大部分属性是很难直接评估的，度量方法都是在统计替代对象的情况。也就是说，度量方法本质上并没有直接评估属性，而是在提供替代对象的信息。基于某种理论或假设，度量者认为替代对象的信息反应了属性的特征，即算法和属性间存在合理的联系。表9-4总结了一些常见度量的属性、替代对象和联系。

表9-4　属性和算法的联系

属　　性	算法所使用的替代对象	联系（算法所依赖的假设）
代码的复杂度	分支和循环关键字的出现频率	● 分支和循环可能带来复杂的执行路径，复杂的执行路径会导致理解困难。 ● 分支和循环越密集，代码就越复杂。 ● 分支和循环所带来的复杂性是代码复杂性的主要来源
产品质量	当前活跃的缺陷数目	● 产品功能已经基本完成，所有未实现的功能都有缺陷在追踪。 ● 测试人员在全力寻找错误,他们的测试策略是多样且有威力的。 ● 测试人员有充分的时间和资源来寻找错误。 ● 测试人员报告了他们发现的所有缺陷和疑虑。 ● 缺陷评审小组认真地评审了所有缺陷，并制定了正确的解决方法（修复、不予修复、延后修复）。 ● 程序能够正确地修复缺陷，不引入更多的修复。 ● 任何人都遵守标准的缺陷处理流程，不会随意关闭缺陷
测试执行的质量	测试执行所覆盖的语句	● 如果测试覆盖了某条语句，那么它就覆盖了该语句对应的所有机器代码。 ● 如果测试覆盖了某条语句，那么它很可能发现其中的缺陷。 ● 如果测试没有覆盖某些语句，那么它不能发现其中的缺陷
测试的进度	记录在案的测试用例数目和已经执行的测试用例数目	● 大部分测试用例拥有相似的粒度和工作量。 ● 大部分测试用例已经记录在案。 ● 测试人员不会再记录更多的测试用例。这意味着软件需求冻结，软件设计不再变更，新的项目风险不会出现。 ● 测试人员会记录每个测试用例的结果。 ● 测试人员在测试过程中不设计新的测试用例。这意味着他不调查测试所发现的新情况
测试人员的工作绩效	测试人员所发现缺陷的数目	● 测试人员的大部分时间都在执行测试并报告缺陷。这意味着他不进行其他技术调查活动，包括帮助其他人进行测试。 ● 测试人员所测试的模块拥有相似的测试复杂度和工作量。 ● 测试人员所测试的模块拥有相当的缺陷数量。 ● 程序员向测试人员提供了相同的技术支持

由表9-4可知，常见的度量方法往往依赖于一组假设。任何稍有工作经验的人都可以看出许多假设并非总是成立。这意味着度量值所提供的信息往往是有偏差的，是不符合真实情况的。而且，度量值通常是对复杂属性的简单评估，即便有“一叶知秋”的潜力，也难免“一叶障目”的风险。所以，有经验的测试人员只把度量方法当做技术调查的一种工具，用它们来获得深入研究的线索，而非提供最终答案。

因为度量结果总是有偏差的、片面的，所以仅凭借数字就作出项目决策是危险的。为了获得全面认识，测试人员会使用多个度量方法从不同角度评估属性，并交叉比对度量值。他会进一步直接研究调查对象，获得第一手资料。以下是一些例子。

❑ 当测试人员分析代码复杂性时，他会仔细阅读复杂度较高的代码，分析它们为什么拥有

较高的度量值。更重要的是，他会针对代码的特点构思相应的测试想法。

- 当测试人员分析测试活动的质量时，他会从多个方面考察测试覆盖的情况：结构、功能、数据、接口、平台、操作、时间等（参见2.2.2节）。基于调查结果，他会补充设计测试用例，以弥补测试覆盖的漏洞。
- 当测试经理评估测试进度时，他会在多个维度上分析已经完成的测试工作：缺陷、需求、代码、配置、变更历史、程序员、测试人员、数据等。任何单一的度量方法都是不够的，且常常产生误导。
- 当产品经理需要了解产品质量时，他会用“卖点漫游”去考察产品最具价值的情景，并用真实的任务来考验它的表现。此外，他会和测试人员交谈，了解他们发现的问题，倾听他们对项目风险和产品缺点的担忧。他还会和程序员交流，了解他们所面临的困难和需要完成的任务。在很多时候，询问团队成员对项目和产品的“感觉”，能够获得许多有价值的信息。
- 当测试经理想要了解下属的工作情况时，他会阅读测试人员提交的缺陷报告，以了解他的测试方法、诊断问题的技巧、书写报告的严谨程度。然后，他会与测试人员面谈，知晓他面对的具体问题，以及他所采取的解决方案。

当测试人员获得度量值之后，他需要自问：这些度量值该作何解释？如何解释度量算法与属性的联系？如何解释度量算法的合理性？我需要具体调查哪些对象来获得解释？为什么度量值如此之低（或如此之高）？它们反应了什么情况？针对这些情况，我应该采取何种行动？这些问题将驱动他做一些后续研究，获得更全面的理解。

当测试人员向管理层提交度量值之前，他必须能够清晰地解释度量值的含义，并用具体的案例来支持自己的观点。如果某个度量值引起了项目领导的兴趣，他会向测试人员询问相关情况。如果测试人员不能解释自己给出的数值，他的专业水平会受到质疑，而职业的测试人员能够阐述度量值所反应的情况，给出详细的案例说明，并提出自己的建议。

9.4.4　理解度量方法的优点和缺点

在掌握了属性和算法的联系后，测试人员需要理解度量方法的优点和缺点。例如，语句覆盖率是常用的测试覆盖率度量方法，它拥有如下优点。

- 粒度适中。从编译器的角度，程序设计语言的基本构建块是常量、标识符和运算符，它们构成了表达式，表达式构成了语句，语句构成了更复杂的函数和类。虽然语句并不是语法分析的最小元素，但是它仍旧是编译器解析代码的基本元素，能够较准确地映射到编译生成的机器代码上。以语句作为覆盖率分析的目标，能够较细致地评估测试执行所覆盖的机器代码。
- 简单易懂。语句通常是编写产品逻辑的基本单元，是代码理解的基本元素。测试人员能够直观地理解语句覆盖率的结果，并针对尚未覆盖的语句产生新的测试想法。

- 方便实用。目前有许多成熟的工具可以评估测试执行的语句覆盖率，并生成精美易读的报告。测试人员只需付出很少的精力去配置工具，就能够获得测试执行的语句覆盖率，为产生更多的测试想法提供信息。

与此同时，语句覆盖率也具有一些不可忽视的不足。

- 编译器通常将程序设计语言的一条语句翻译为多条机器指令。有时，某条语句被测试覆盖并不意味着所有的机器指令被测试覆盖。例如，语句`bool x = condition1() && condition2();`对两个函数的返回值实施与运算。如果函数`condition1`返回`false`，那么与运算符`&&`不会调用函数`condition2`，它会立即给出计算结果`false`。有些覆盖率工具不能标记出`condition2`没有被执行，它们会笼统地记录语句`bool x = condition1() && condition2();`被测试覆盖。可见，虽然语句覆盖是一种基于结构的覆盖，统计语句覆盖率的工具可能会漏掉一些程序结构。
- 作为一种基于结构的覆盖方法，语句覆盖率不能发现那些应该被实现但是没有被实现的代码。许多缺陷的根源是程序员没有为特殊情况或异常情况编写处理代码，语句覆盖率对发现此类错误没有帮助。此外，另一些缺陷的原因是代码不能合理地处理输入数据或程序状态，语句覆盖对发现此类数据与状态也没有帮助。这意味着，100%的语句覆盖并不能保证程序经受了足够的测试。
- 测试人员需要考虑结构、功能、数据、接口、平台、操作、时间等覆盖因素。如果测试人员只专注于语句覆盖（结构覆盖），那么他会错失许多有价值的测试想法，从而遗漏一些严重的缺陷。

严格地说，任何度量方法都是对属性的简化，其结果都丢失了属性的一些重要信息。因此，测试人员必须掌握方法的不足，唯有如此他才能在恰当的范围中运用方法，并规避潜在的问题。

9.4.5　密切关注度量的副作用

在许多项目中，项目管理者尝试用度量值来推动项目发展，并激励项目人员在相关方面做得更好。这种良好的意图可能导致意料之外的副作用。当员工意识到管理层在强制要求度量值到达某种水平时，他们会改变自己的行为，使得度量值更“好看”，而不是让产品变得更好。恰如前两节所讨论的，度量值只有在特定假设成立时才能反应部分的真实情况，而强制追求度量值的压力会破坏有效度量的前提，使得员工不能正常工作，领导不能获得真实的信息。

许多团队会用缺陷数目来评估产品质量。由表9-4可知，该度量方法的有效性依赖于一系列前提条件。不幸的是，这些前提很容易被破坏。9.1.2节提到了一个真实的案例，在产品发布前夕，项目的管理者制定了一条规则：任何一个新缺陷必须在24小时内解决。他们期望程序员和测试人员可以通力合作，让产品代码能够快速稳定下来。然而，这条基于缺陷数目的规则导致了一些不良副作用。以下是对当时情况的描述。



对于一些复杂的缺陷，没有人可以承诺在24小时内完成修复和检验。但是，程序员和测试人员仍旧愿意修复它们。因为产品是他们的工作成果，体现了他们的价值，所以他们想竭尽所能去提高其质量。不幸的是，缺陷评审小组会因为缺陷无法在24小时内修复，将其解决为“不予修复”或“延迟修复”。这挫伤了第一线员工的士气。

为了让缺陷得到修复，测试人员在发现缺陷之后，并不提交正式的缺陷报告，而是直接通知程序员，让其着手修复。在程序员在完成修复后，测试人员才正式提交缺陷。也就是说，测试人员和程序员达成了协议，共同开发出一个秘密的开发流程。他们一边在维护管理层所希望的度量值，一边在按自己的心愿来改进产品。

另一种让缺陷得到修复的办法是只提交最严重的缺陷，以获得缺陷评审小组的修复许可。在签入代码时，程序员会同时签入多个缺陷的修复。也就是说，程序员和测试人员保存了一份不公开的缺陷列表，他们会批量地编写、测试和签入修复代码。

无论哪种方法都使得管理层不能准确地知晓目前有哪些活跃的缺陷、缺陷的严重程度如何、缺陷发现呈现出何种趋势，也使得缺陷报告不能反应缺陷修复的工作量。这动摇了项目决策的数据基础，令项目发展承担了更大的风险。令人惋惜的是，管理者、程序员、测试人员都在以自己的方式来帮助产品，却令产品质量受到威胁。

有些项目团队不但统计缺陷数目，还会统计测试用例总数和已通过的测试用例数目，以评估测试活动的进度。我认为这种基于测试用例数目的进度度量不会获得期望的效果，且具有严重的副作用。

- 测试人员无法预测有多少测试用例会被执行。在测试过程中，新发现会激发新的测试想法，进而生成更多的测试用例。因为测试人员不能预料会发现哪些信息和缺陷，他也不能给出最终的测试用例总数。既然估算测试用例总数是“不可能的任务”，测试人员就只能提供一个不确定性很大的估算值。利用这样的估算值评估项目情况是有风险的。
- 强制要求测试人员记录所有测试用例及其执行结果会浪费测试人员的时间。测试是一个探索过程，测试人员设计测试用例，以刺探产品信息，并根据产品的反应设计更多的测试用例。记录所有的测试用例及其结果会打断“设计→实验→反思→再设计”的迭代过程，降低了测试效率。而且，并不是所有的测试用例都有文档化的价值。许多测试用例只需要运行一次，将它们记录在案只是纯粹的文案工作，不会为项目提供价值。在大多数情况下，测试人员只需记录测试数据、测试策略（一组指导测试设计的测试想法）和所发现的缺陷与风险（为后续的测试设计提供指导信息）。更多讨论请参考3.2.13节介绍的测程表。
- 因为详细地预测和记录测试用例不是切实可行的测试活动，测试人员通常不会认真地统计测试用例总数和已通过的测试用例个数（即便他们认真对待，也会受挫）。这意味着他们提供的信息是不准确的，而使用不准确的信息会导致错误的项目决定。此外，不合理的度量会挫伤员工的积极性，隐性地降低工作效率。

坦率地说，度量的副作用往往来自管理层制定的开发流程。测试人员可能没有勇气公开对项目管理者提出反对意见。然而，沉默并不会缓解问题，负责任的测试人员应该做正确的事。这并不容易，也没有通用的应对方法。本节提供两点建议，供测试人员参考。

第一，测试人员应该坚持原则，以正直的态度来服务团队。这体现为测试人员应该提供诚实的信息，包括如实提交缺陷、报告进度、提供度量值等。即便存在“湮灭”缺陷的压力，测试人员也要如实提交所发现的错误。即便度量方法不甚合理，测试人员也要提供真实而非杜撰的数值。诚实地提供技术信息是测试人员的职责，也是后续改进的基础。

第二，测试人员应该密切关注度量的副作用。如果发现某个度量正在产生负面影响，他应该构建理论、收集证据和提出改进方法。也就是说，他要能解释该度量的基本元素（意图、属性、算法、联系），并阐述它会在当前项目环境中产生不良副作用。然后，他需要收集支持其观点的案例和数据。仅仅表明当前度量存在问题是不够的，测试人员还需要思考如何改进当前流程，提出相应的解决方法。之后，他应该与测试经理一对一地面谈，解释自己的理论、证据和新方法，向他提供第一线员工对工作流程的反馈。有时，测试人员可以和几个同事达成一致意见，然后一起向测试经理反馈意见，利用团队的力量来增强观点的说服力。

9.4.6　注重实效的计算

与9.3.3节所介绍的历史数据收集一样，度量的计算也应该力求简单、自动和准确。

首先，测试人员应该优先考虑简单的度量方法，如代码行数统计、缺陷个数统计等。这些度量方法拥有明确的统计对象，简单明了的算法，能够更直接地阐述度量值的含义。度量只是技术调查的辅助工具，简单的方法能用较小的付出获得足够好的信息，为后续调查留下更多的资源和时间。随着项目的发展，他可以逐渐引入一些新的度量方法。这是一个实验的过程，他会保留被实践证明行之有效的方法，并去除那些没有帮助的方法。

其次，测试人员应该尽量自动地计算度量结果。本书的7.1.1节、8.2.1节、8.2.2节、9.3.3节对此提供了一些建议。简而言之，测试人员应该掌握一门开发效率高的程序语言，从而快速地实现度量算法。考虑到许多团队用数据库存储项目数据，熟练地使用SQL将帮助测试人员极大地提高开发效率。此外，电子表格软件能快速地完成一些计算任务，且提供了强大的作图功能，也是测试人员计算和呈现度量值的好工具。

第三，测试人员应该在测试活动结束后立即收集度量所需要的输入数据。准确的度量值依赖于正确的算法和准确的数据。为了避免记忆错误、数据丢失等问题，测试人员应该尽早收集必要的项目数据，并妥善保存。如果用测程组织测试活动，他可以在一个测程结束后，记录相关数据。此外，他也可以在每天下班前回顾当天的工作，反思测试策略，总结测试进度，并记录必要的数据。

9.5 测试小组

测试小组是项目团队的一部分，是一个小型的团队。所谓团队不是一群在同一地点工作的人，而是一群彼此支持以达成共同目标的人。本节讨论测试人员如何加入测试小组的团队协作中。

9.5.1 价值观

适应团队风格的第一步是了解团队的价值观。有些团队会明确阐述自己所秉持的价值观，并在日常工作中予以贯彻。即便没有明文记载，测试人员仍旧可以感受到测试小组固有的工作风格和价值取舍。在这样的团队中，测试人员要遵循团队的价值观，使自己的行为符合同事与领导的期望。

不过，有些团队并没有明确的价值观，或者测试小组的行为并不像测试经理所宣称的那样。判断测试经理（或测试人员）是否真的重视某件事，就要看他有没有投入时间和精力去把它做好。如果测试经理宣称他非常重视测试人员的协作，却很少出席例行的测试协调会议，那么测试人员就知道“测试协作”并不是他最重视的东西。如果测试经理反复强调自动化测试的重要性，测试小组却任由大量的测试用例遭遇失败而不予修复，那么测试人员就知道“自动化测试的稳定性”并不是高优先级的任务。无论团队风格如何，测试人员都必须对自己负责，他需要确定自己的价值观，以引导自己的职业发展。

不同的测试人员基于自身情况和项目环境会建立不同的价值观。其中，有两点是需要坚持的基础。第一点是本章所反复强调的“正直”，这是一个信息提供者的信誉基础。第二点是“互助”，这是团队工作的基础。

互助的第一步是高质量地完成自己的工作。一方面，只有先行完成自己的工作，才有额外的时间去帮助测试伙伴。另一方面，测试人员之间是相互配合的关系，高质量的工作成果可以支持其他测试人员更好地工作。

对于测试人员而言，互助还意味着在团队中分享测试情报，让其他人可以了解最新情况，避开已知问题，从而更有效地探索软件。例如，测试人员可以在以下领域中作出贡献。

- **测试小组周会**。许多测试小组会举办周会，以颁布制度、通报情况、布置任务、协调活动、展示技术等。通常，测试经理会主持周会，负责具体任务的测试人员会介绍相应的情况。如果测试人员有信息需要分享给测试小组，他可以事先通知测试经理，以便将相关内容纳入会议议程。在会议上，他会介绍情况，并回答小组成员的提问。如果有一些问题不能当场解决，他应该妥善记录，并在会后积极跟进。
- **测试小组知识库**。组织良好的测试小组都会考虑构建自己的知识库。知识库的载体通常是支持多人协作编辑的Wiki网页或共享笔记本（如Micorosft OneNote），其内容涉及测试活动的方方面面，包括测试制度、工作流程、测试指南、测试想法列表、检查列表、移交文档、缺陷模板、文档模板、常用工具、推荐实践等。好的知识库提供了一个众所周

知的地点，让测试小组分享其知识与经验，让测试人员方便地搜索有用的信息。测试人员应该积极地向知识库贡献信息，将工作中发现的、学到的、感悟的知识记录下来。这些信息不但能帮助测试小组，而且从长远看，对于测试人员的职业发展有涓滴成河之效。经过一段时间的持续积累，测试人员会发现当初记录的片段信息渐渐构成了体系，自身的能力也有长足的进步。

- **定期的测试报告**。恰如Cem Kaner等测试专家所指出的，测试小组的力量来源于沟通，而定期的状态报告是传递信息的强有力的工具[Kaner01]。因此，测试人员应该定期向测试经理报告工作情况。如果他承担了一项服务于测试小组的任务，他也应该定期向测试小组报告任务进展。一般而言，状态报告的默认频率是每周一次。对于一些不紧急的任务，可以两、三周报告一次，最长间隔不要超过一个月。在项目临近发布时，他可能需要每天报告，以方便测试小组掌握进度、协调工作。

团队的价值观最终体现为个体的实际行动。测试人员的行为不但能帮助他融入团队，也会对团队的价值观产生影响。坚持原则并积极互助能帮助测试人员和测试小组更好地发展。

9.5.2　团队建设

在互助的基础上，测试人员需要积极参与团队建设的活动。所谓团队是为了解决共同问题而紧密协作的一组人，所以团队建设的有效方法是让小组成员一起完成任务。这里有两个关键点：第一，小组成员需要协作才能获得成功，单枪匹马的作战无法取得胜利；第二，任务需要成功完成，胜利会帮助一群人凝聚为团队。基于这两点，以下活动对于团队建设有潜在的帮助。

- **集成测试**。在集成测试或系统测试中，测试小组的成员会测试同一个产品实例，他们要相互配合以完成彼此依赖的测试任务。在此过程中，他们会一起制订计划，共同解决阻碍测试的问题，交换彼此拥有的信息，携手完成测试流程。有经验的测试领导和测试人员会努力组织集成测试活动，通过密切的协调和交流，让集成测试和团队建设都获得成功。
- **结对测试、组队测试和头脑风暴会议**。这些是7.3.3节所介绍的团队测试活动，其特点都是多个测试人员共同测试一个产品实例。在短时间内，他们密切地交流，相互传授业务知识和测试技能，以激发出大量的、多样化的测试想法。这些测试活动充分利用了测试人员的差异性和互补性，通过实际测试展示出协同作战要胜过单枪匹马的战斗。此外，它们也具有团队学习的特征，能让好的测试技术在测试小组中得到传播。
- **缺陷清扫**。与结对测试、组队测试和头脑风暴会议相似，缺陷清扫是一项短期的全员测试活动。除了测试人员，有些缺陷清扫还邀请程序员、产品经理、内部用户等共同参与。其主要优势在于引入了“更多的眼睛”：程序员更了解代码实现，有可能发现隐藏的缺陷；测试人员更擅长缺陷猜测和持续攻击，有可能发现被其他测试人员所遗漏的缺陷；程序经理更理解领域和业务，有可能发现流程和设计上的缺陷；内部用户是软件的使用者，有可能发现易用性上的缺陷。总之，参与者在技能和角色上的差异性有助于发现不同类型的缺陷。

接下来，本节将介绍一个缺陷清扫的真实案例，它反应了一些团队建设的基本要点。当时，我所在的测试小组正在测试一个在线产品。在发布前，我们进行了一次缺陷清扫，其组织与执行过程如下。



测试小组大约有20名测试员工。每个员工会负责一个或几个子系统的测试，这些子系统会组成整个在线系统。在集成测试阶段，测试领导组织了一次缺陷清扫，以再次探测子系统协作中的缺陷。

测试经理提前一周公布了测试安排：测试时间是下周三下午1点到5点，测试地点是一个足够容纳所有小组成员的会议室，每位员工携带笔记本电脑前往测试。此外，他还列举了此次测试需要重点关注的子系统和使用情景，并提供了缺陷模板。

随后，测试经理安排几位员工去搭建被测试系统。他们选择了几台计算机作为缺陷清扫的“专用测试机”，部署了所有子系统，导入了来自产品环境的业务数据。接着，他们做了一些基本的端到端测试，确保该系统不存在阻碍深入测试的问题。

至此，缺陷清扫的准备工作完成。回顾准备阶段，不难看出高效的缺陷清扫需要精心的安排。首先，测试组织者要明确地通报测试安排，邀请测试人员参与测试。第二，他需要公布测试指南。这并不是束缚测试人员的思路，而是强调本次测试的重点，以确保测试小组确实测试过相关领域。第三，测试组织者要提供必要的测试环境，包括足够舒适的会议室、足够稳定的被测系统等。

在缺陷清扫当天，测试小组成员进入会议室，一起测试。会议室中有一张很大的方桌（由多张桌子拼接而成），让所有人可以围桌而坐。如此安排座位的好处是，测试人员一抬头就可以看到其他人的脸，能够方便地对话。当测试人员测试他人负责的子系统时，他可能不了解该子系统的业务目标和使用方式。这时，他可以询问子系统的测试负责人，以立即获得解答。这种“即时通信”使得测试可以顺畅地推进，也自然构建了测试人员之间的互助关系。

在日常工作中，测试人员大多在独立工作，彼此之间的联系并不紧密。在缺陷清扫中，测试人员进行渗透式交流，能够较紧密地协作。有时，一位测试人员分享他发现的严重缺陷，这激起了其他人的测试灵感，从而能挖掘出更多的问题。有时，他们一起嘲笑拙劣的设计和滑稽的缺陷，甚至说一些笑话以相互逗乐。这种活跃的气氛有助于更积极、更自由地测试。

经过一段时间，测试小组发现了一些个人测试难以发现的缺陷。首先，测试小组覆盖了整个线上系统，执行了许多长流程、跨子系统的情景，暴露出一些集成层面的问题。第二，测试人员在测试技巧和思考方式上存在差异性，当他们测试他人的系统时，能够发现一些被遗漏的缺陷。

作为测试组织者，测试经理亲自参与缺陷清扫。他也坐在方桌边一起工作。他的行为让测试小组意识到领导对缺陷清扫和团队协作的重视，因为他切实投入了自己的时

间。在测试过程中，测试经理每小时都会通报测试进度：目前已经发现了多少缺陷、谁提交了比较多的缺陷、哪些缺陷是值得注意的等。利用缺陷数目，他用实在的测试进展鼓舞了测试小组的士气，也推动了测试人员之间的良性竞争。

当测试进行到一半的时候，测试经理提供了一些食品，包括薯片、饼干、水果、饮料等。于是，测试人员暂时放下测试，休息片刻。他们一边吃东西，一边闲聊。这样的“中场休息”有利于测试人员在“下半场”继续工作。

在缺陷清扫结束后，测试经理统计所提交的缺陷，然后发送总结报告给测试小组。该报告感谢小组成员的参与，罗列出所有缺陷，并公布提交缺陷最多的前三名。几天后，测试经理向这三名员工颁奖，奖品是餐券、书券等小礼品。奖品并不贵重，其目的是提高整个活动的趣味性。

构建高效团队并没有什么一定奏效的方法，经理所能做的是营造一个恰当的环境，让团队逐渐地自行凝聚[DeMacro99]。在此过程中，测试人员应该有意识地帮助测试伙伴和测试经理，这对他的职业发展也大有裨益。

9.6　小结

本章讨论了测试人员如何有效地服务于团队，并实施恰当的测试管理。

- 测试人员通过服务团队来体现自己的价值。这要求他设定正确的目标，高质量地完成工作，并乐于助人。
- 为了对自己负责，测试人员应该建立正确的价值观。其中，正直和互助是核心价值。
- 测试人员的影响力来自于出色的工作成果和工作中所表现出的能力。
- 团队成员都在尽力帮助项目，测试人员应该支持他们，并用技术调查检验其工作。
- 个人测试计划是项目测试计划的延伸，需要考虑各种项目元素，并聚焦风险。
- 测试活动需要动态管理，通过测试迭代、动态优先级、有效估算、及时报告等实践来适应不断变化的项目语境。
- 测试人员需要经常自问：“我现在可以测试什么？能够如何测试？”
- 测试人员应该估算自己的任务，以此为依据拟定计划，然后做出承诺。
- 计数和计算是估算的基本手段，项目数据是它们的处理对象。因此，准确估算的第一步是妥善地收集项目数据。
- 同时估算最差情况和最好情况是常用的估算技巧。而定期比较估算值和实际值，分析不准确估算的原因，也有助于提高估算能力。
- 掌握度量方法，需要了解它的意图、属性、算法、联系、副作用。
- 所有度量方法都具有局限性，要了解它们的不足，并避免它们的负面影响。
- 凝聚团队需要小组成员通力协作，成功完成一个单人无法承担的任务。
- “重视”一件事应该体现为花时间和精力把它做好。

第10章

个人管理

第9章指出测试人员是执行经理，需要积极地管理自己的时间和工作。执行经理同时也是专业人员，应该坚持专业主义，追求精湛的技艺和卓越的工作。为了对团队和自己负责，他应该主动实施个人管理，通过每天的努力来积极推动其职业发展。

本章讨论个人管理的基本实践，包含时间管理、个人学习、专业发展等内容。这些主题与个人特点紧密相关，所以不同的人会发展出不同风格的方法，且都能获得很好的效果。本章将结合我的实践，介绍一些常见的原则和方法，供测试人员参考。

10.1 时间管理

在激烈的市场竞争中，测试工作呈现出进度紧、任务多的情形，且经常被突发事件打扰。为了应对这些挑战，测试人员需要主动管理自己的时间，因为时间是最重要的项目资源之一，它深远地影响了测试活动的目标设定、策略选择和执行方式。积极的时间管理将促成更好的测试过程，提高个人和团队的绩效。

时间管理本质上是任务管理。它意味着为一组任务制订计划，然后积极地行动，使重要的任务可以在期望的时间内完成。对于测试人员而言，时间管理是测试管理的一部分，其目的是在项目期限内完成一组动态变化的测试任务。因此，第9章所讨论的测试计划、测试迭代、动态管理、软件估算等内容都可以应用于时间管理。本节将介绍我是如何具体运用这些方法的。

10.1.1 利用任务清单记录所有工作项

对我而言，时间管理的第一步是维护一份任务清单，用它记录并追踪本周的工作项。这份清单既是时间管理信息的“数据库”，存储了所有的工作项信息，同时也是一份工作计划，安排了任务的执行顺序。这样做的主要意图是专注、备忘、计划和追踪。

- 高效工作的第一步是保持专注。这意味着暂时“忘记”其他任务，集中精力把当前任务做好。为了避免“永久忘记”而错过完成期限，我需要把待办事项都记录下来。这让我

更放心地工作，因为我知道它们已经被妥善保存，可以随时查阅。此外，如果我强迫自己去记忆这些工作项，难免要在头脑中反复回顾，这无形中破坏了专注的氛围，既分散精力，又顾此失彼。

- 在一周的工作中，我需要完成多个工作项，而且每天都可能接到新任务。为了做到“要事第一”，我需要比较这些工作项的轻重，并为它们安排恰当的时间和资源。因此，我需要将它们记录下来，以随时审视当前的工作项，动态地安排优先级，并持续追踪完成进度。

我试过几种任务清单的形式。一开始，我将工作项记录在一张A4纸上，完成一项就将它划掉。写满一张纸，就换一张白纸，并将未做完的工作项誊写上去。该方法简单易用，任务清单一直躺在工作桌上，触手可及，随时待查。当新任务出现时，我只需在纸上记上一行，就可以继续当前工作，几乎不产生中断开销。

后来，我听从了《时间管理——给系统管理员》（*Time Management for System Administrators*）一书的建议：把所有信息集中在一个地方，以便随时随地访问 [Limoncelli05]。于是，我将工作项记录到记事本中。记事本的优点是可以随身携带，有任何想法都可以及时记录。我带着它参加各种会议，如果发现新的信息和任务，我会记录下来，并在会后将新任务补充到本周工作列表中。下班后，如果我对第二天的工作产生了新想法，我会立即将它写入记事本，从而避免遗忘。

现在，我将Microsoft OneNote[①]作为任务清单的载体，并将笔记存放在Office 365[②]的服务器上。电子笔记本较纸质笔记本更容易修改，没有誊写的开销。而且，随着云计算的发展，电子笔记本能够存储在云端，使用计算机和智能手机都可以访问，同样具有随时随地读写的能力。我依然使用记事本记录日常工作中发现的信息和产生的想法，但是会将工作项都转移到电子笔记本中，使它们在一个地方被存储和管理。

图10-1是我的工作清单样例，它由3个部分组成。第一部分是本周的工作项列表。在使用电子笔记本之后，我会为较复杂的工作项建立单独的页面，并用链接连接工作项和页面。该页面是对应任务的工作簿，用于记录测试过程的各种信息。它没有固定的格式，其内容可能包含被测构建的版本号、构建位置、参考资料链接、测试环境配置、测试想法、检查列表、测试发现等。将任务清单和任务页面放在同一个笔记本内，让我在测试时可以快速地查阅信息，并方便地记录测试笔记。

在实际工作中，任务页面容纳了测试信息、测试策略和测试日志，是多种测试文档的综合体。以前，我习惯用纸笔来记录工作项和测试笔记，但是我发现这常常导致信息的分散。例如，测试经理发送电子邮件来通知新任务，我将该任务写入记事本。在测试时，我经常要参考任务邮件，以获得必要的测试信息。这强迫我在记事本（测试笔记）和邮件（测试信息）之间来回切换，无形中耗费了精力。现在，我可以直接将邮件内容粘贴到任务页面中，使得参考信息随时可用。而且，电子笔记本支持截图、标签、列表、富文本等对象，令测试笔记更加生动。在某种意义上，该笔记本为

---

① http://www.onenote.com。

② http://www.office365.com。

专心工作提供了一个“工作空间”，使得测试活动的信息输入和输出无需在多个信息源间切换。此外，我会定期存档任务页面。今后若接到相似的任务，我会找出这些页面，以获得参考资料。

任务清单的第二部分是本周日程表，它为工作项安排了具体的工作日。这要求我估算工作项的时间，并根据优先级等因素安排其先后顺序。我将性能测试安排在周一，是为了尽早提供性能测试报告，让程序员有更多的时间去调查性能问题。集成测试被放在周二和周三，是因为它需要大约1.5天的工作量，而且我希望尽早产生业务数据供其他测试小组使用。缺陷清扫的团队活动预定在周三举行，所以该工作项只能置于周三。我将补丁测试安排在周四和周五，是因为没有其他测试活动依赖于它，只要在本周完成即可。

日程安排的本质是测试计划，即拟定恰当的方案使得工作项（以最大的概率）在一周内完成，并提供符合关系人期望的成果。通过日程安排，我可以判断本周能否完成预定的任务。如果时间不够完成所有工作项，我会标记出一些可以转移到下周的任务，将它们的优先级降到最低（通常安排到周四、周五）。如果时间还是不够，我会和测试经理讨论，共同想出一个可行的方案。有关个人计划和工作量估算的更详细讨论请参考9.2节和9.3节。

任务清单的第三部分是今日工作安排，它以列表的方式呈现了当天的具体工作。图10-1呈现了周一的工作安排，其中有一项工作是“修订本周计划。这是因为测试小组周会有时会分配一些新的测试任务，我会根据会议内容重新调整计划。

图10-1　任务清单

任务清单是一份活动的工作计划，它反应了我对当前任务的安排。下一步我要做的是根据工作进展调整任务清单，使工作计划不是过时文案，而是反映真实情况并指引方向的地图。

10.1.2　坚持周计划和每日回顾

由图10-1所示的任务清单不难看出，我在实施周循环所要求的周计划。所谓周循环就是每周选定工作项，估算时间并制定方案，然后尽全力去完成任务。敏捷开发专家Kent Beck指出周循环是极限编程的基本实践之一，是软件人员开始敏捷旅途的可行起点 [Beck04]。在以往的几个项目中，我都实施周计划，并获得了不错的效果。因此，9.2节也建议测试人员使用周迭代作为个人测试计划的基本单元，以此来实现测试活动的动态管理。

为了追踪周计划的实施情况，我实施每日回顾。在下班前，我会检查当日工作列表（如图10-1所示任务清单的第三部分），并制定隔天的工作列表。其主要活动如下。

- 为新出现的任务安排时间。在项目过程中，我经常接到一些计划外的任务，或想到一些值得尝试的测试策略。我会将这些工作项都列入周计划，然后在每日回顾中调整日程安排（如图10-1所示任务清单的第二部分），为它们分配工作日。
- 将当日未完成的工作项移到第二天。有时，一些任务的用时超出了估算，令一些工作项不得不延后。每日回顾帮助我了解工作的真实进展，避免“埋头苦干”却忘记了实际进度。在工作项移动的过程中，我会特别关注高优先级任务，评估它们不能完成的风险。如果某项本周必须完成的工作很可能要延期，我将与测试经理讨论相关的解决方案。
- 为第二天安排具体的活动。这类似于“微型计划”，将工作项分解为一系列小的活动，并确定相应目标。这些具体的活动和目标将指引我一步步去完成工作项。在获得具体的工作安排后，我会安心回家，因为我知道第二天的工作已经“尽在掌握”。

周一是特殊的一天，它的工作列表是根据本周计划所制订的。周一上班后，我会拟订一份周计划，其依据是上周的工作情况和当前项目进度。然后，我参加测试小组周会，收集更多的项目信息。会后，我根据新信息调整周计划，并确定周一的工作列表。对我而言，周一上午是制订周计划的时间，而周一的工作列表是周计划的一部分。

每日回顾将周循环和日循环连接在一起，使任务清单（工作计划）总是反应最新的工作进展。及时的调整不但帮助我更灵活地分配工作时间，也增强了我完成任务的信心，降低了“最后期限”带来的压力。

10.1.3　专注是高效工作的前提

对于完成任务而言，计划只是第一步，更重要的是有效的执行。执行任务往往会依赖于外部资源和内部动力。从时间管理的角度看，专注的工作可能是最重要的内部动力。

许多时间管理方法建议工作者分配一段时间，在其中排除干扰，专心致志地完成一件任务。例如，Jonthan Bach和James Bach提出了基于测程的测试管理，建议用一个时间盒（时长为60~120分钟）测试一个主题 [Bach00a][Bach04a]。软件专家Staffan Noteberg在介绍番茄工作法时，建议程序员将工作分解为一系列番茄钟[Noteberg10]。一个番茄钟是一个时间盒（通常是25分钟），用于完成一个明确的任务。如果任务没有完成，就再分配一个番茄钟。知名博客作家Leo Babauta也强调聚焦一个目标的单任务模式[Babauta09]。他的工作方式是每天早上做的第一件事是全天最重要的事情。在做完之前，绝不碰其他事情。做完之后，他会稍作休息，然后做下一件重要的事情。还有一些类似的方法，其核心都是划出一段时间做一件事情，在"与世隔绝"的状态中，尽全力工作。

在日常工作中，我会从当天的工作安排中选择一个任务，根据其内容分配一段时间，然后在该时间盒内一心一意地工作。对于小型测试任务，时间盒一般是15~30分钟。对于大型测试任务，时间盒一般是60分钟。如果60分钟不能完成任务，我会稍作休息，然后分配一个30~60分钟的时间盒，以继续工作。我曾经连续3个小时持续编码，不作任何休息（其实3个小时是保守估计，因为我过于投入，已经忘记任务何时开始和结束），也获得了很好的工作成果。但是，我觉得长时间连续工作并不适用于大多数测试与开发任务。通常，持续工作超过60分钟就会导致注意力分散和效率下降。此时，应该离开办公桌，喝一点水，走动一圈，让头脑和身体都得到放松。张弛有度才能在工作时集中精力。

除了拥有专属时间，专注工作还需要避免干扰。当我打算专心工作时，我会关闭邮件客户端、即时通信软件和其他无关软件，以避免分散精力在电子邮件、即时消息、网络冲浪上。有些工作信息来自电子邮件，我会将此类信息都复制到电子笔记本中。在工作时，我只参考电子笔记本，并始终关闭邮件客户端，这样就不会被新邮件所打扰。基本思路是只开启必要的软件，构建一个"自给自足"的工作空间，以抵御其他活动的诱惑。

一些测试人员觉得不能关闭邮件客户端和即时通信软件，这样会导致他们不能立即响应紧急情况。对于某些项目的特定角色而言，确实如此，立即响应特定事件是他们的职责。但是，对于大多数项目，测试人员"离线"几个小时，并不会导致项目灾难。相反，专注工作能更快地提供更好的工作成果，反而有利于项目快速推进。

另一个提高工作效率的小技巧是充分利用多显示器。编程和测试都是高强度的知识处理活动，人脑需要摄入、处理、产出大量的信息。这些信息源自和流向不同的信息仓库，它们在计算机屏幕上体现为一组分离的窗口。为了让信息传递更加流畅，测试人员应该降低信息流转的阻力，即降低窗口切换的开销。

图10-2是我的工作环境，由3台显示器组成。在编程时，左侧的显示器承载了编程接口的帮助文档，中间的显示器展示了代码编辑窗口，右侧的显示器则提供额外的代码窗口和命令行。在测试时，左侧的显示器呈现电子笔记本，以展示测试想法并记录测试笔记；中间的显示器安放被

测产品的窗口，以支持测试执行；右侧的显示器拥有一组调试和诊断工具，以监控被测产品。这样的显示器安排让我快速访问各种窗口，令信息流转更加流畅。

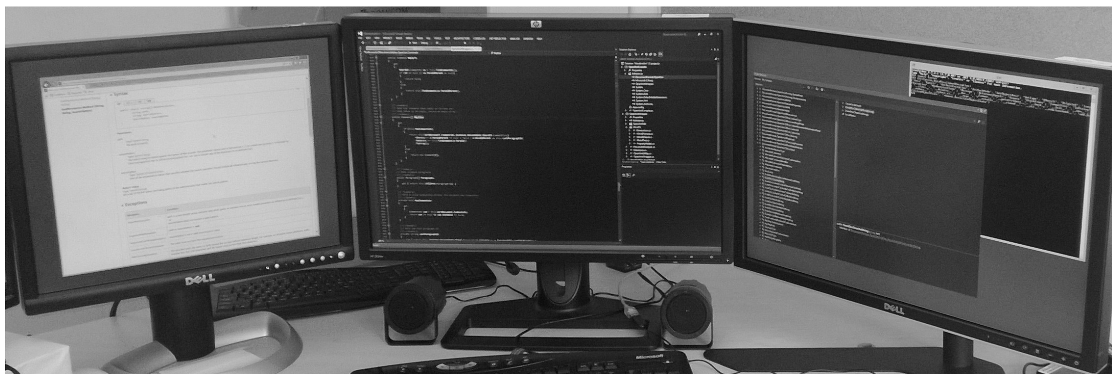


图10-2　利用多显示器来提高工作效率

除了测试人员的个人努力，测试小组也可以有意识地创造“安静时间”[Ford08]，以提高团队效率。我的项目团队有一条会议规则：所有的例行会议都应该安排在周一到周四的上午。这样，所有的团队成员（项目领导、产品经理、程序员、测试人员）都拥有无会议的周一至周四的下午和周五全天。虽然有些临时性的会议难免被安排在下午和周五，但是程序员和测试人员都拥有大块时间可以自由分配（产品经理出于工作需要，会组织和参加许多会议）。这样可以降低任务中断的概率，帮助团队成员更自由地使用时间。可见，追求高效的团队应该主动安排团队时间，让个人能够专注地工作。

10.1.4　恰到好处的文档化和自动化

关于节省时间，我有两点经验。

第一，犯错会浪费时间，认真把事情做对能节省时间。有时，我为了赶上进度会匆忙工作，然而仓促的过程往往导致一些错误。为了修正错误，我不得不花费大量时间进行故障诊断和返工，这是一些任务超时的根本原因。经过几次教训，我渐渐领悟到“欲速则不达”的道理。唯有高质量的工作才能获得稳定的速度，才能在进度压力下完成紧迫的任务，任何以降低质量为代价的“捷径”都会导致失败。

第二，参考成功的解决方案有助于把事情做对，从而节省时间。如果某个方法经过实践考验，能够有效解决某个问题，那么复用或模仿它很可能再次成功。然而，人的记忆力是不可靠的，单凭模糊的记忆去操作很可能导致错误，进而浪费时间在故障调查和问题修复上。为了保存一些重要的信息，我会将它们写入个人的电子笔记本或测试小组的知识库（其载体也是电子笔记本）。通常，我用个人笔记本保存一些私人的想法，将分享给整个小组的信息写入知识库。

在多数情况下，我不会预先编制大量的文档，而是根据当前任务做必要的文案。即便是记录一些通用的知识，结合具体任务来陈述能获得更好的效果。以下是一些常见的文档编写时机。

- 一边测试，一边记录。一般，我会在测试之前列出一批初始的测试想法，从大方向上思考测试类型和测试策略。在实际测试中，测试反馈会提供丰富的信息，能激发出许多测试想法，有些想法并不奏效，有些想法却极具威力。我会记录那些能够提高测试覆盖、对缺陷有较强检测能力的想法，为今后的测试提供支持。
- 在发现错误时，记录正确的做法。这里的错误不是软件缺陷，而是我所犯下的差错。软件测试是非常复杂的技术调查，我常常犯下错误，导致缺陷遗漏、进度延长、时间浪费。为了避免我自己和同事再犯相似的错误，我会记录正确的做法或需要检查的要点，以节省未来的工作时间。
- 当解除疑惑时，记录答案。在工作中，我经常遇到一些产品疑问和技术难题，有时我会独立调查，有时我会寻求同事的帮助。相似的，同事们也会向我咨询一些我了解的问题。在获得或提供答案之后，我会考虑该答案是否值得记录以备复用。如果更多的人也会遇到该问题，或该答案提供了一个有价值的解决方案，我会把它写入知识库，供整个小组参考。
- 在集体测试活动之前，编写测试指南。一些测试活动需要测试小组成员相互协作才能完成，为此活动负责人应该提供必要的测试信息，分享给小组。以缺陷清扫（参见9.5.2节）为例，测试负责人会要求各模块负责人提交简短的测试说明，以陈述各模块有哪些潜在风险、用户情景、测试重点和已知问题。之后，他将汇总后的结果发布给测试小组，帮助他们更好地理解测试任务。

在我看来，文档工作的秘诀是“恰到好处”，最忌讳“以辞害意”。也就是说，应该用简洁的语言、恰当的形式记录最核心的内容，以求写得轻快、读得方便。反之，生搬硬套模板，盲目追求形式，只会让作者和读者受挫。以下是一些文档编写的小技巧。

- 用操作文档（参见3.2.10节）来呈现如何完成一个复杂的任务。该文档由一系列步骤组成，每个步骤用文字描述、代码片段、数据文件、屏幕截图等形式讲述特定操作。其核心价值在于舍弃了所有修饰，针对特定任务直截了当地给出解决方案，使得任何人都“按图索骥”地完成工作。
- 用测试想法列表（参见3.2.8节）或检查列表（参见3.2.11节）来记录测试策略。在本质上，测试策略是一组指导测试设计的想法，因此列表能够自然地呈现测试策略的多样性和可延伸性。对于测试人员而言，列表能快速传达有益的测试提示，又不会约束他的自由发挥。而且，其简洁、开放的形式能鼓励他用很小的代价增加更多的测试想法。经过几轮测试，列表会多方面地覆盖被测领域，为周密测试提供有益参考。
- 使用代码片段、屏幕截图、视频录像、超级链接等元素简化文档编写。软件测试是一项技术性很强的工作，涉及许多细节，有些内容很难用文字清晰地描述。对此，我选择直

接记录必要的细节。如果是阐述代码设计，我会考虑提供核心代码的片段，并高亮出关键语句。如果是讲解操作细节，我会考虑提供屏幕截图，并勾勒出重点内容。如果是讲解一个软件的用法，我会考虑录制一个10分钟的操作录像，以讲解常用操作。如果所讲述的概念比较复杂，我会给出参考文献的超级链接，让读者获得更多资料。图10-3是我在2011年利用业余时间测试必应词典桌面版的笔记[1]。它既是一份测试日志，也是一份常用调试工具的演示教程。在图10-3中，屏幕截图展示了技术细节，图片上的标记勾勒出重点，超级链接指向更详细的信息，文字旁边的问号点出重要情况，高亮文字标识出我想强调的内容。可见，有效地利用富媒体的表现手段，能用很少的文字传达大量的信息，明显提高了文档编写的效率。

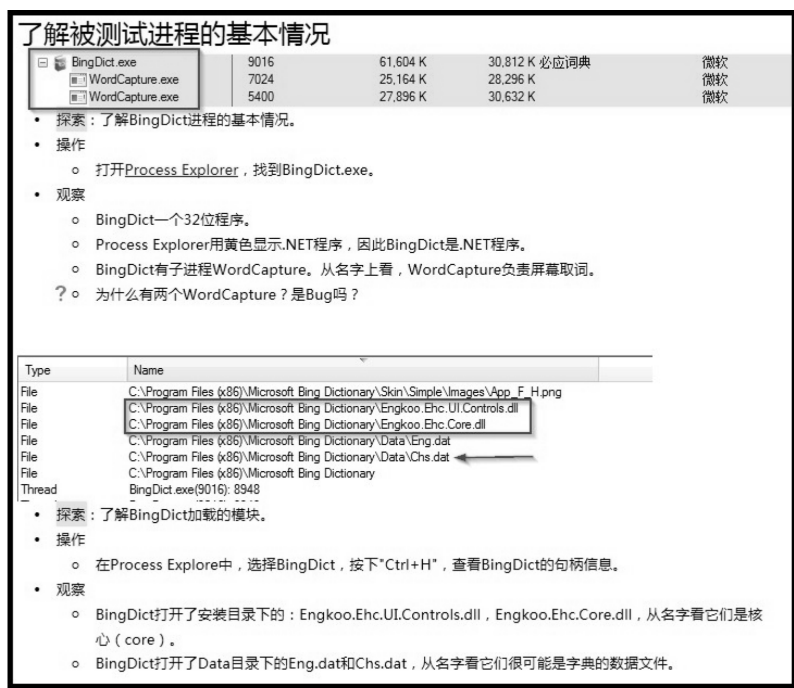


图10-3　用屏幕截图来简化编写并丰富文档

自动化代码是一种特殊的技术文档，用计算机可理解的方式表达了解决方法。其优势不仅在于自动执行能够节省人力，还在于代码是明确、具体、无歧义的描述，能够精确地传递知识。代码编写者以外的人在理解这些知识之后，可以发展出更具威力的解决方法，或将现有策略快速地移植到其他领域。

测试人员没有时间把所有任务都自动化，而且也不是所有的事情都适合自动化。Thomas A.

---

① 在线地址：https://skydrive.live.com/view.aspx/Public/exploratory^_testing?cid=da204d6ee69a08b2&id=documents。

Limoncelli将系统管理员需要应对的问题分成4类，并给出了相应的自动化见解[Limoncelli05]。我认为他的观点也适用于产品安装、补丁安装、测试环境配置、测试环境更新、数据准备、产品操控、结果分析等测试任务。

- 只做一次的简单事情。此类任务通常无需自动化，因为编写并调试代码会花费更多的时间。不过，如果有空闲的时间且自动化开销不大，我还是会编写自动化代码。一方面，这是一种刻意的练习，让我复习编程接口和程序语言。另一方面，有时很难确保该任务“只此一次”，轻便的自动化开销不大，并节省了未来的时间。以我的经验，完成核心步骤的代码拥有较高的复用率。
- 只做一次的困难事情。我会将该任务分解为一些小任务，自动化每个小任务，再将它们串联起来，获得完整的自动化方案。有时，某个小任务不适合自动化，我会混合手工操作和自动化执行来完成整个任务。
- 常做的简单事情。此类任务最适合自动化，而且自动化的投入很快就能获得回报。6.3.2节介绍了相关案例，其特征都是编写短小的代码，以快速完成软件控制、环境配置、数据分析等任务。有些手工执行只需十几秒的操作序列，只要是机械的、重复的就值得自动化。代码清单10-1展示了一段AutoHotKey脚本，它监听到快捷键Win+F2后，会发送键盘命令Ctrl+C，将当前选中的文字复制到剪贴板，然后在两个搜索引擎上查询剪贴板内容。这段脚本每次只能节省几秒的时间，但是它让我摆脱了机械操作（复制文字、启动浏览器、输入网址、粘贴文字并搜索、输入第二个网址、再粘贴文字并搜索），将精力专注在核心任务上。
- 常做的困难事情。此类任务往往具有复杂的流程，需要处理多种可能情况。对此，我会与测试经理协商，安排一段专用时间来完成自动化。解决方案通常是基于现有软件系统做二次开发，使之可以处理新的流程。如果项目进度不允许花费大量时间用于自动化，我会编写操作文档或检查列表，帮助手工及半自动操作顺利完成。

代码清单10-1　调用多个搜索引擎的AutoHotKey代码

```
#F2::                                                ; 快捷键是Win+F2
Send ^c                                              ; 发送Ctrl+C
Run http://www.bing.com/search?q=%clipboard%         ; 在必应上搜索剪贴板内容
Run http://www.google.com/search?q=%clipboard%       ; 在谷歌上搜索剪贴板内容
return                                               ; 命令结束
```

通常，我会用DOS Shell、PowerShell、IronPython、AutoHotKey等脚本语言来完成小型自动化，用C#和SQL来构建较复杂的自动化方案。我的感受是，自动化代码积累得越多，测试效率就越高。一方面，持续地编写代码令我更熟悉程序语言和编程接口，能够更快地开发新代码。另一方面，已有代码是很好的知识库，通过代码复用和简单改写，我可以立即解决当前环境的常见问题。在某种意义上，有价值的代码是“活的”代码，它会被复用，并在复用过程中被不断增强，从而发展出灵活的结构和强大的能力。

10.2　持续学习

软件行业总在高速发展中，测试人员为了胜任工作和推动职业发展，必须持续地学习。所谓学习是“求学”（获取知识）和“练习”（应用知识）的组合，是吸纳、实践、反思的循环过程。也就是说，学习并非单纯地摄取知识，还需要积极地实践，通过反馈和反思来巩固学习成果。基本上，来自实践的知识只有在实践之后才能真正掌握。

10.2.1　在工作中学习

为了高效地完成测试任务，测试人员需学习许多内容，包括领域知识、用户情景、测试技术、计算平台、开发技术、软件项目、项目团队等。面对如此多的内容，一条值得参考的指导原则是，确定项目团队或所在领域最需要的技能，然后努力掌握它们 [Kaner01]。对于此类知识，通过实际工作来掌握是一种比较好的学习方法。这样做可以加速获取知识与应用知识的循环，并让学习成果立即帮助测试过程。成功的应用能够增强测试人员的信心，鼓舞他更深入地研究。

实际上，本书的大部分内容都在讨论如何学习产品和项目。第2章到第5章讨论了使用多种测试方法实施技术调查，第7章介绍了产品研究，第8章探讨了项目研究。在此，本节只简单地介绍当我接手一个功能时，会做哪些初步的学习。

- 首先，我会实施积极阅读（7.3.2节），评审该功能的移交文档（3.2.14节），以掌握该功能的基本情况，包括需求与功能、质量标准、基本测试想法、现有自动化测试、可用测试数据、已知缺陷、潜在风险等。然后，我与原测试负责人讨论这些内容，了解他测试该功能的经验与技巧，并请教在文档阅读过程中产生的疑惑。
- 除了团队内部文档，我还会参考外部资料。例如，在测试Web应用时，我会在网络上搜索Web开发和测试的资料，了解基本编程技术、常用测试策略、典型缺陷和可用工具，并评估在当前项目中如何运用这些信息。又例如，在测试Microsoft Office产品时，我会收集一些专著和网络文献，阅读相关内容。经过多年的使用和研究，很多熟练用户已经成长为Office专家，他们撰写了大量的书籍和资料。阅读这些文献可以了解用户最常使用、专家最常推荐的使用情景，并发掘出许多内部文档没有记录的细节。
- 更重要的是，我会通过漫游测试（5.4.4节）和快速测试（5.5节）来实际操作软件，建立或完善该功能的一批测试模型（4.2节）。常见的测试结果是功能列表（3.2.4节）、测试想法列表（3.2.8节）、输入与输出模型（4.2.2节）、系统生态图（4.2.3节）和更新了的移交文档等。

随着测试的发展，我会进一步研究产品和项目元素，并尝试更多的测试方法。可见，在测试过程中，测试人员不但能学到许多与产品相关的知识，还可以循序渐进地学习行业知识、开发技术、测试方法、测试模型等内容。只要他保持学习的积极性，勇于进入新领域并尝试新技术，测试工作就能帮助他持续地成长。

10.2.2　持续阅读

除了在工作中学习，测试人员还需要超出公司业务和软件测试，拓展自己的职业发展[Kaner01]。专业的测试人员不但为公司工作，还应对自己负责，学习各种知识来提升自身的能力。对于我而言，这意味着阅读大量的书籍。在博客上，我提供了一份阅读清单[①]，列出了40多本我读过的测试书籍。实际上，我的阅读量远不止这些。图10-4展示了我读过的一些软件开发书籍，左上角是我曾经的办公室藏书，右下角是目前的家庭藏书。书籍内容涉及开发技术、测试技术、开发方法论、软件工具、行业故事等，既有经典名著，又有时效性较强的工具书。



图10-4　我读过的软件开发书籍（局部）

对于比较重要的书，我会一边阅读一边做笔记。图10-5展示了我对《完美软件——对软件测试的各种幻想》[Weinberg09]做的笔记，包括用即时贴标记重点页面，用铅笔勾勒重点文字、进行批注和绘制草图。在我看来，笔记的形式和内容并不重要，关键是做笔记可以让我读得更慢一些，并随时记录阅读感想。这将自己的知识和经验带入阅读，让我与书的作者进行虚拟“对话”，使阅读不是单纯的摄取，还伴随着反思。

对于一些工具书，我一般不会全部读完，只是放在案头备查。例如，虽然我经常参考《SQL Server 2005范例代码查询辞典》[②]，但是从没有完整地读过一遍。如果我发现某些内容在工作中被反复参考，我会阅读它们所在的一章，以了解该子领域。通常，我不会阅读与实际任务无关的章节，毕竟在时间有限的情况下，应该将精力集中在有帮助的内容上。无重点的阅读降低了学习效率，也使得需要认真研究的内容没有获得足够的资源。

① http://www.cnblogs.com/liangshi/p/3274750.html。

② http://book.douban.com/subject/1762921/。

阅读需要大量的时间，然而忙碌的工作常常会大幅压缩空闲时间。对此，我采用两个办法来坚持阅读。第一，对于重要的书，我会安排一个固定的时段，每天读一个小时。一个小时并不是严格的约束，空闲时间充裕就多读一会儿，空闲时间紧张就少读一会儿。关键在于每天不间断地阅读，经过一段时间的积累，自然能够获得长足的进展。第二，我会利用旅途、等待、工作间隙等碎片时间来阅读。例如，我曾经坐地铁上班，往返需要一个多小时。于是，我将这一个小时视为额外的阅读时间，总是携带一本书在地铁上浏览。这样每日积累，一个月可以多读一两本书。

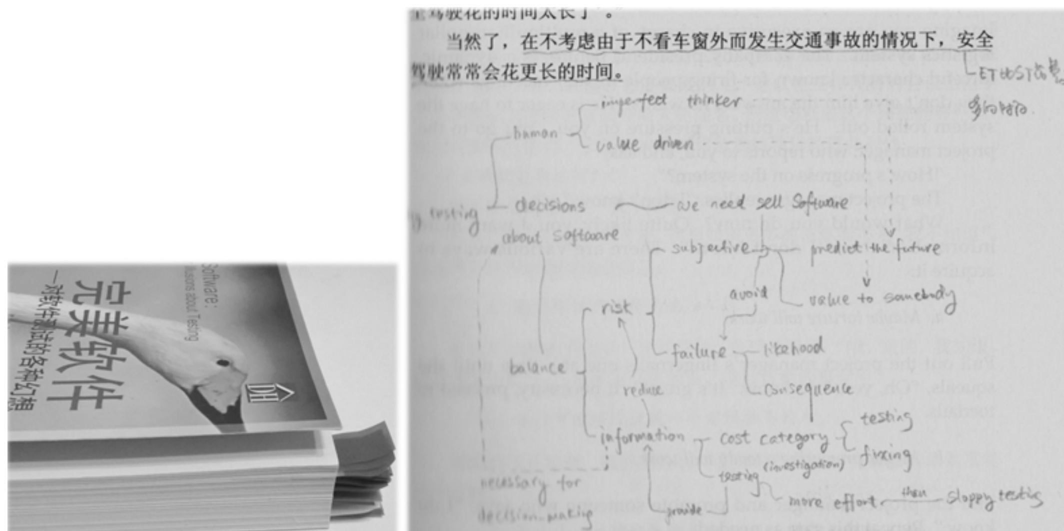


图10-5　我对《完美软件——对软件测试的各种幻想》所做的笔记

除了书籍，我还会经常阅读技术博客和杂志文章。我的切身体会是大量阅读能够拓展知识面，并逐渐建立自己的知识体系。当接触到一本新书或新文章时，我可以快速地评估其内容属于知识体系的哪一部分，它的哪些部分对我有益或无用，它的哪些观点与我一致或相反。这帮助我更有效地阅读文献，并运用其知识。

单纯的阅读并没有完成学与习的循环，还需要实践才能真正掌握知识。对于读到的测试方法，我会刻意在测试工作中运用，通过实践来评估其适用范围。对于读到的编程技术，我会用它编写有实际用途的程序或自动化测试代码。并不是所有方法都适合我的项目语境和个人情况。但是在束之高阁之前，应该花一些时间来实践它们。这可以扩展眼界，体会多种流派、思想和技术，还可以让一些看似不合常理却有效的方法脱颖而出。

10.3　且行且思

经历和经验是两个紧密相关又相互区别的概念：经历是一段工作的过程，而经验是从该过程中学习到的理论和方法。从个人努力的角度，经历只与工作时间和工作内容有关，它会随着工作

年限而自然增长，但是经验则需要主动地积累和反思。也就是说，“两年的工作经历”并不会自动转化成“两年的工作经验”，测试人员能力的成长取决于他如何对待和反思其实践。而且，如果他不能及时发现工作中的坏习惯，更长的工作时间只会强化不良实践，经历愈长愈有害。

关于经验积累与分享，软件开发者Dave Hoover和Adewale Oshineye在其著作《软件开发者路线图：从学徒到高手》（*Apprenticeship Patterns: Guidance for the Aspiring Software Craftsman*）中提出了一些很好的方法 [Hoover09]。其中，且行且思、记录所学和分享所学也是我一直坚持的实践。

“且行且思”这个言简意赅又意味深长的表述来自王江平先生传神的翻译。一方面，它暗示软件人员的职业生涯是“漫漫长路”，需要坚持不懈地跋涉；另一方面，它指出持续的努力应该伴随持续的思考，执行与反思缺一不可。具体而言，专业人员需要建立对自己工作的反馈回路，通过定期审视其工作，识别出成功的实践和失败的实践，并制定相应的改进方案。

在工作中，我将“且行且思”和“记录所学”结合在一起。所谓“记录所学”就是持续地记录自己学习到的知识和感悟到的想法。和许多开发者一样，我同时使用两种记录载体，一个是私人工作日志，用于保存个人的想法，一个是团队知识库，用于分享对团队有价值的知识。这意味着我可以利用两种反馈回路，个人记录帮助我诚实面对自己的进步与错误，公开记录则将经验传递给团队并接受更广泛的反馈。

图10-6展示了我的私人工作日志。这是一个A4幅面的记事本，用黑色的签字笔书写并绘制内容。它没有严格的格式，大致以列表的形式罗列了想法。

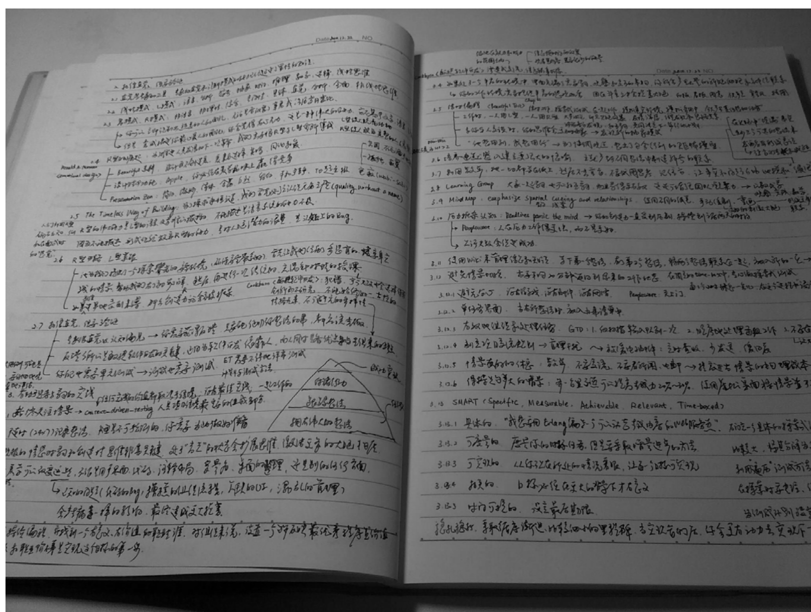


图10-6　私人工作日志

通常，我会在周末回顾一周的工作（且行且思），并将工作经历和个人感悟写入工作日志（记录所学）。在撰写时，我通常会从以下方向开始思考。

- 我测试了哪些功能？使用了什么测试策略？为什么使用这些策略？哪些策略发挥了比较好的效果，应该坚持使用？哪些策略效果不佳，需要改进？
- 我开发了哪些代码？进行了哪些设计决策？这些决策的依据是什么？哪些决策发挥了比较好的效果？哪些决策效果不佳？这对今后的开发工作有哪些启示？
- 我参与了哪些协作性活动？我的哪些行为推动了协作？有没有更好的协作方法？我的哪些行为不利于协作？我应该如何改进？
- 我做对了什么？该做法为什么有效？它坚持了什么价值观，应用了何种实践方法？未来应该如何坚持？
- 我做错了什么？错误的根源是什么？我应该如何改进？
- 我观察到什么有趣的事情或现象？这对我有什么启发？
- 我有哪些有趣的想法？

我对日志内容没有严格的限定，只是宽泛地记录本周的所行所想。因为仅供自己参考，所以我从不担心文字表达是否条理清晰，尽管以自己能够理解的方式不断地书写下去。即便某些页面因为修改文字和补充想法显得有些凌乱，我也不会在意。有时，这些改动和补充恰恰忠实记录了某个想法的演变过程。

软件大师Jerry Weinbeg是个人日志的长期实践者，他道出了日志的一个重要优点："不像书籍或课程，日志所记录的一切都与你有关。因为个人学习都是私人的，我不能保证你能学到什么，但是我可以保证你一定会学到一些有价值的东西。"[Bolton07]我的切身感受也是如此，日志记录了我的所学所想，提供了稳定且持续的自我反馈。除了每周反思和记录，我会定期读一读以前写的日志，用新的眼光来审视过去的想法和决策，用已有的案例和经验来考察现在的情况。该积极的复读过程可以梳理我的工作经历，并将分离的想法有机地联系在一起，构成自己的价值观和知识体系。在撰写本书的过程中，我也多次参考现有的工作日志（共5本），不但获得了很多资料，也再次反思了自己的工作方法。

通过"且行且思"和"记录所学"可以积累许多经验，这时应该"分享所学"。在日常工作中，我会将知识和经验分享到团队知识库，或通过邮件、报告等形式广播给整个小组。此外，我还会选择一些有价值且不涉及机密的内容，撰写成文章，发布在我的博客上[①]。

在我看来，撰写博客是很好的个人实践。首先，它是一种学习方法，推动我更周详地分析要发表的内容，从而获得只有深入思考才能得到的洞察力和系统性。当我自认为已经了解一个主题时，有条理地阐述它会暴露出我原有认识的不足。为了更好地论述，我会做必要的研究，包括阅读文献和动手实验。从这个角度看，撰写过程本质是研究过程，博客文章是研究过程的自然结果。

① 我的博客网址：http://cnblogs.com/liangshi/。

第二，技术写作是一个专业人员应该具备的素质，它对书面交流和技术传播非常重要。提高写作能力的核心方法就是多写，撰写博客为此提供了很好的练习机会。第三，发表博客文章可以从更广大的社区中获得反馈，并“以文会友”去认识一些志同道合的朋友。例如，我和高翔通过彼此的博客发现双方都对探索式测试有浓厚的兴趣，于是经常交换意见和分享经验，自然成为好友。后来，我们一起合作撰写了《探索式测试实践之路》[①]一书，将所学所知分享给整个中文测试社区。第四，博客按时间顺序记录了作者的工作历程。经过一段时间的积累，它不但成为作者的个人知识库，还是他的个人“名片”，能够有说服力地展示其技术能力和思考水平，帮助他获得社区的认可。

在本质上，且行且思、记录所学和分享所学的核心是持续的反思和积累。这要求测试人员建立反馈回路，通过各种途径来评估自己的行为和方法，并记录下经验教训和改进方案。其功效并非一朝一夕就可显现，但是集腋成裘的积累会获得丰硕的成果。

10.4　成为专家

在测试职业发展上，测试人员有两条路线：成为技术专家或团队领导。在更大的范围上，他可以选择新的研发角色，如程序员或产品经理，甚至离开软件行业，投身新的领域。不管选择哪条道路，他都应该主动设定目标，并积极进取。本节将简单讨论测试技术专家这条职业路线。

首先，测试人员应该成为一名通才。软件本身的复杂性及其应用的广泛性要求测试人员从多个关系人的角度，综合运用多种方法实施技术调查。这意味着测试人员需要超出当前任务范围去掌握大量的测试方法。唯有如此，当他切入新领域或接到新任务时，才能根据语境开发出有针对性的、注重实效的测试策略。

恰如测试专家所说，“不要幻想只要两个星期就可以成为柔道黑带”[Kaner01]，成为通才需要艰苦的努力和漫长的积累。“书山有路勤为径，学海无涯苦作舟”——坚持不懈是职业发展中一个重要的（也可能是最重要的）策略。此外，一个有帮助的方法是先了解一些框架性方法，以大致掌握测试领域的脉络。例如，六要素测试分类系统（参见5.1节）是测试方法的分类框架，启发式测试策略模型（参见4.2.1节）是测试设计的参考框架，启发式测试计划的语境模型（参见3.2.1节）是测试计划的参考框架，基于测程的测试管理（参见3.2.13节）是测试执行的框架。在建立了理论框架之后，测试人员可以在工作与学习中不停地累积具体的技术，并将“血肉”添加到“骨架”之中。久而久之，其知识体系自然能够“枝繁叶茂”。

一名测试新人需要避免的思维误区是用某个职务头衔限制自身能力的发展。例如，我在测试论坛中发现一些测试人员常讨论“白盒测试工程师”、“黑盒测试工程师”和“性能测试工程师”等职位。一部分人认为它们是相互排斥的，即做白盒工程师就不用考虑用户情景，做黑盒工程师

① 该书主页：http://cnblogs.com/etbook/。

就不用考虑代码实现，做性能工程师就只要钻研性能测试工具。其实，这些名词只是某些公司所设定的职位而已，只代表他们对工程师的要求，并不体现软件行业对高水平测试人员的期望。实际上，国内外的高水平科技企业都要求工程师能够独当一面，能够独立完成一个子领域的大部分任务。对于测试人员而言，他需要能够独立完成一个子系统的测试，无论测试活动是白盒测试、黑盒测试还是性能测试，只要工作需要，他就应该有能力完成。因此，为了长远地发展职业生涯，测试人员不应该被头衔所约束，而是要积极地拓展自己的技术广度。

在广泛涉猎的基础上，测试人员需要选定一个领域，成为该领域的专家。这意味着要了解该领域的概念、理论、技术和细节，能够提出高层的设计方案，也能解决困难的具体问题。当团队成员遇到困难时会主动向他咨询，他能够快速地给出答案。如果遇到某个非常困难的问题，不能立即回答，他经过一段时间的研究，能够清晰地解释问题的根源，并提供相应的缓解方案。有时，他会给出若干方案，并分析每种方案的优缺点，然后自然导出目前的最佳方案。可见，专家是该领域的咨询师，其声望与职务头衔无关，而是来自于帮助团队解决实际问题的能力。

在工作中，我接触过一些专家。虽然他们有不同的工作风格，但是仍表现出一些共性的特征，为测试人员的发展提供了参考范例。

- 专家透彻地了解他的领域。在软件测试中，有一些领域需要较强的能力和长期的积累，如行业知识、情景测试、安全测试、国际化测试、性能测试、测试开发、调试诊断等。专家会在特定的一个或几个领域中长期投入，积累丰富的技术和经验。面对具体问题时，他们能够提供合理的解决方案，或给出实用的建议。
- 专家会持续拓展其专业领域。在高速发展的软件行业中，他们会根据项目和职业发展的需要切入新领域，并成为新领域的专家。过去的经验和成功并不会将他们束缚在原有的领域中，相反它们为持续探索提供了动力和基础。
- 专家会主动承担领导责任。技术专家一般不承担领导职务，但他们从来都是团队的技术领导。其领导力不是来自于任命，而是来自于强烈的责任心、渊博的知识、丰富的经验和出色的工作，因此它拥有牢固的基础和天然的说服力。在测试小组中，专家会组织具体活动，编写相应的测试指南，回答测试人员的问题，并给出专业的建议。例如，安全测试专家会编写安全测试指南，并在团队知识库中记录安全测试相关的技能和工具。在项目中，他评审所有系统和组件的测试设计，从安全测试的角度给出有价值的建议。此外，他会定期组织针对安全缺陷的集体测试活动，如结对测试、组队测试、头脑风暴会议（参见7.3.3节）等。在此过程中，他与小组成员一起测试，现场演示方法和工具，讲解其原理和技巧，帮助他人更好地实施安全测试。
- 专家会主动将自己的工作和知识分享给团队。专家拥有良好的书面表达和口语表达能力，能通过团队知识库、技术报告、电子邮件、专题讲座、口头交流等形式传递知识。此外，他会主动选择并开始一个技术项目，在取得切实进展后，将阶段性成果报告给测试小组。这些成果能够帮助测试人员更好地工作，也鼓励他们加入进来，推动项目的持续发展。

不难看出，以上实践并没有超越任务管理、持续学习、知行合一、且行且思、记录所学、分享所学的范畴，只是专家的努力付出获得了令人赞叹的成绩。这表明测试人员遵循相同的原则和方向，通过研究、实践和反思，也可以成长为专家。这是一条漫漫长路，但是耐心和坚持能帮助测试人员稳步向上。

10.5 小结

本章讨论了如何有效利用时间来促进职业发展，介绍了一些常见的实践方法。

- 时间管理的本质是任务管理，即为一组任务制订合理的计划，然后有效地执行，并动态地调整计划。
- 应该把所有任务信息集中在一个地方，以便随时参考和修改。
- 周计划和每日回顾有助于实施动态计划，以适应复杂且变化的项目情况。
- 高效工作的第一步是保持专注。
- 测试人员应该主动创造专注的工作环境，“单任务”的时间盒是常见方法。
- 错误会浪费大量的时间，认真把事情做对能节省时间。
- 恰到好处的文档化和自动化能够用较小的代价呈现切实可行的解决方案，有助于避免错误，节省时间。
- 学习的本质是“知行合一”，是吸纳、实践、反思的循环过程。
- 通过工作来学习项目或领域最需要的技能，有事半功倍之效。
- 测试人员需要超出公司业务和软件测试，来扩大自己的知识储备。对此，坚持阅读是一个基本方法。
- 为了从实践中获得更多，测试人员要且行且思、记录所学、分享所学。
- 测试人员应该同时拓展其技术能力的广度和深度，成为通才和专家。
- 坚持不懈是职业发展的基础性策略。

参考文献

[Adler72] Adler M J, Van Doren C. *How to Read a Book: The Classic Guide to Intelligent Reading*. Touchstone, 1972.

[Agans06] Agans DJ. *Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems*. AMACOM, 2006.

[Babauta09] Babauta L. *The Power of Less: The Fine Art of Limiting Yourself to the Essential...in Business and in Life*. Hyperion, 2009.

[Bach00a] Bach J. Satisfice Test Context Model. http://www.satisfice.com/tools/satisfice-cm.pdf, 2000.

[Bach00b] Bach J. Session Based Test Management. http://www.satisfice.com/articles/sbtm.pdf, 2000.

[Bach04a] Bach J. Testing in Session - How to measure exploratory testing. http://www.sasqag.org/pastmeetings/ExploratoryTesting_SessionBasedTestManagement.pdf, 2004.

[Bach04b] Bach J, Schroeder P J. Pairwise Testing: A Best Practice That Isn't. http://www.testingeducation.org/wtst5/PairwisePNSQC2004.pdf, 2004.

[Bach09] Bach J, Bach J. ET with Subtitles. http://www.youtube.com/watch?v=Vy0I2SB5OLo, 2009.

[Bach10] Bach J. Telling Your Exploratory Story. http://agile2010.agilealliance.org/files/Telling%20Your%20Exploratory%20Story%20Agile2010.pdf, Agile 2010 Conference, 2010.

[Bach11] Bach J, Michael Bolton. Rapid Software Testing. Satisfice Inc., http://www.satisfice.com/rst.pdf,2011.

[Bach12] Bach J. Heuristic Test Strategy Model. http://www.satisfice.com/tools/htsm.pdf, 2012.

[Bach99] Bach J. Test Plan Building Process. http://www.satisfice.com/tools/build-the-plan.pdf, 1999.

[Beck04] Beck K, Andres C. *Extreme Programming Explained: Embrace Change, 2nd Edition*. Addison-Wesley, 2004.

[Bender07] Bender R. The Ambiguity Review Process. http://www.benderrbt.com/Ambiguityprocess.pdf, 2007.

[Blamurugadas13] Blamurugadas Ajay. Context-free Questions Mindmap. http://enjoytesting.blogspot.com/2013/02/context-free-questions-mindmap.html, 2013.

[Bolon10] Bolton M. Context-Free Questions for Testing. http://www.developsense.com/blog/2010/11/context-free-questions-for-testing/, 2010.

[Bolton05] Bolton M. Testing Without a Map. http://www.scribd.com/doc/47080783/2005-01-Testing-WithoutAMap, Better Software, 2005.

[Bolton07] Bolton M. An Exploratory Tester's Notebook.http://www.developsense.com/presentations/etnotebook.pdf, 2007.

[Bolton09a] Bolton M. Of Testing Tours and Dashboards.http://www.developsense.com/blog/2009/04/of-testing-tours-and-dashboards/, 2009.

[Bolton09b] Bolton M. An Exploratory Tester's Notebook. http://www.developsense.com/presentations/etnotebook.pdf, 2009.

[Bolton12] Bolton M. FEW HICCUPPS. http://www.developsense.com/blog/2012/07/few-hiccupps/, 2012.

[Buwalda04] Buwalda H. Soap Opera Testing. Better Software, www.stickyminds.com, http://www.logigear.com/logi_media_dir/Documents/Soap_Opera_Testing.pdf, 2004.

[Carvalho11] Carvalho P. Lessons Learned in Session-Based Exploratory Testing. http://staqs.com/docs/sbtm/, 2011.

[Carvalho2011] Carvalho P. Lessons Learned in Session-Based Exploratory Testing. http://staqs.com/docs/sbtm/, 2011.

[Chess07] Chess B, West J. *Secure Programming with Static Analysis*. Addison-Wesley, 2007.

[Cohn04] Cohn M. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, 2004.

[Cohn09] Cohn M. The Forgotten Layer of the Test Automation Pyramid. http://blog.mountaingoatsoftware.com/the-forgotten-layer-of-the-test-automation-pyramid, 2009.

[ContextDrvienTesting12] Kaner C, Bach J. Context Driven Testing. http://context-driven-testing.com/, 2012.

[Crispin09] Crispin L, Gregory J. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 2009.

[DeMarco03] Demarco T, Lister T. *Waltzing With Bears: Managing Risk on Software Projects*. Dorset House, 2003.

[DeMarco99] DeMarco T, Lister T. *Peopleware: Productive Projects and Teams*. Dorset House, 1999.

[DeMarco08] Demarco T, Hruschka P, Lister Tim, etal : *Understanding Patterns of Project Behavior*. Dorset House, 2008.

[Drucker93] Drucker P F. *The Effective Executive: The Definitive Guide to Getting the Right Things Done*. Harperbusiness, 1993.

[Duvall07] Duvall P M , Matyas S, Glover A. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional, 2007.

[Edgren11] Edgren R. The Little Black Book on Test Design. http://thetesteye.com/blog/2011/09/the-little-black-book-on-test-design/, 2011.

[Edgren12] Edgren R. Many Models - Better Test Ideas. http://thetesteye.com/blog/2012/02/many-models-better-test-ideas/, 2012.

[Feathers07] Feathers M. *Working Effectively with Legacy Code*. Addison-Wesley, 2007.

[Ford08] Ford N. *The Productive Programmer*. O'Reilly Media, 2008.

[Gallagher12] Gallagher T. 新版Office的安全工程和产品改进. http://www.csdn.net/article/a/2012-12-13/2812771, 2012.

[Gause89] Gause D C, Weinberg G. *Exploring Requirements: Quality before Design*. Dorset House Publishing Company, 1989.

[Gof94] Gamma E, Helm R, Johnson R, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[Hendrickson13] Hendrickson E. *Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing*. The Pragmatic Bookshelf, 2013.

[Hilo12] Microsoft patterns & practices. Developing an end-to-end Windows Store app using C++ and XAML: Hilo (Windows). http://msdn.microsoft.com/en-us/library/windows/apps/jj160318.aspx, 2012.

[Hoover09] Hoover D, Oshineye A. *Apprenticeship Patterns: Guidance for the Aspiring Software Craftsman*. O'Reilly Media, 2009.

[Hunter10] Hunter M J. You Are Note Done Yet: Checklist. http://www.thebraidytester.com/downloads/YouAreNotDoneYet.pdf, 2010.

[Johnson10] Johnson K N. *Beautiful Testing*. O'Reilly Media, 2010.

[Kaner00] Kaner C. Rethinking Software Metrics. Software Testing & Quality Engineering, http://www.kaner.com/pdfs/rethinking_sw_metrics.pdf, 2000.

[Kaner01] Kaner C, Bach J, Pettichord B. *Lessons Learned in Software Testing*. Wiley, 2001.

[Kaner06] Kaner C, Exploratory Testing. http://www.testingeducation.org/

[Kaner07] Kaner C, Bach J. Exploratory Testing in Pairs. http://www.kaner.com/pdfs/exptest.pdf, 2007.

[Kaner08] Kaner C, Bach J. Black Box Software Testing: Bug Advocacy. http://www.testingeducation.org/BBST/bugadvocacy/BugAdvocacy2008.pdf, 2008.

[Kaner08a] Kaner C, Bach J. Advocacy B. http://www.testingeducation.org/BBST/bugadvocacy/Bug-Advocacy2008.pdf, BBST Testing Course, 2008.

[Kaner08b] Kaner C. The Value of Checklists and the Danger of Scripts: What Legal Training Suggests for Testers. http://www.kaner.com/pdfs/ValueOfChecklists.pdf, Conference of the Association for Software Testing, 2008.

[Kaner08c] Kaner C. A Tutorial in Exploratory Testing. http://www.kaner.com/pdfs/QAIExploring.pdf, QUEST 2008.

[Kaner10] Kaner C, Bach J. Black Box Software Testing: Foundations. http://www.testingeducation.org/BBST/foundations/BBSTFoundationsNov2010.pdf, 2010.

[Kaner11] Kaner C, Fiedler R L. Black Box Software Testing: Test Design - A Survey of Black Box Software Testing Techniques. http://www.testingeducation.org/BBST/testdesign/, BBST Testing Course, 2011.

[Kaner12] Kaner C, Bach J. The Seven Basic Principles of the Context-Driven School. http://www.

context-driven-testing.com, 2012.
[Kaner13] Kaner C. An Overview of High Volume Automated Testing. http://kaner.com/?p=278, 2013.
[Kaner99] Kaner C, Falk J, Nguyen H Q. *Testing Computer Software, 2nd Edition*. Wiley, 1999.
[Kelly05a] Kelly M. Coming up with a heuristic.http://michaeldkelly.com/blog/2005/8/19/coming-up-with-a-heuristic.html,2005
[Kelly05b] Kelly M. Tour Heuristic. http://michaeldkelly.com/blog/2005/9/20/touring-heuristic.html, 2005.
[Kelly11a] Kelly M. Start with a blank sheet of paper. http://www.quicktestingtips.com/tips/2011/11/start-with-a-blank-sheet-of-paper/, 2011.
[Kelly11b] Kelly M. Use Notepad, not Word when you start writing your plans. http://www. quicktestingtips.com/tips/2011/11/use-notepad-not-word-when-you-start-writing-your-plans/, 2011.
[Kohl07] Kohl J. Man and Machine. http://www.kohl.ca/2007/update-new-article-published-man-and-machine/, 2007.
[Kohl12] Kohl J. Tap Into Mobile Application Testing. https://leanpub.com/testmobileapps, 2012.
[Limoncelli05] Limoncelli T A. *Time Management for System Adimistrators*. O'Reilly Media, 2005.
[McConnell06] McConnell S. *Software Estimation: Demystifying the Black Art*. Microsoft Press, 2006.
[McMillan11] McMillan D. Mind Mapping 101. http://www.bettertesting.co.uk/content/?p=956, 2011.
[Meszaros07] Meszaros G. *xUnit Test Patterns: Refactoring Test Code*. Addison- Wesley, 2007.
[Mugridge05] Mugridge R, Cunningham W. *Fit for Developing Software: Framework for Integrated Tests*. Prentice Hall, 2005.
[Myers79] Myers G J. *The Art of Software Testing, Second Edition*. Wiley, 1979.
[Noteberg10]S Noteberg . *Pomodoro Technique Illustrated: Can You Focus - Really Focus - for 25 Minutes?* Pragmatic Bookshelf, 2010.
[NUnitQuickStart12] NUnit team. NUnit Quick Start. http://nunit.org/index.php?p=quickStart&r=2.6.2, 2012.
[QualityTree06] Quality Tree Software. Test Heuristics Cheat Sheet: Data Type Attacks & Web Tests. http://www.quality-testing.com/wp-content/uploads/2009/06/testheuristicscheatsheetv1.pdf, 2006.
[Robbins07] Robbins J. *Debugging Microsoft .NET 2.0 Application*. Microsoft Press, 2007.
[Robinson10] Robinson H. Exploratory Test Automation. http://www.harryrobinson.net/ExploratoryTest-Automation-CAST.pdf, CAST, 2010.
[Spolsky02] Spolsky J.The Law of Leaky Abstractions. http://www.joelonsoftware.com/articles/LeakyAbstractions.html, 2002.
[Sundaram12] Sundaram J K, Needamangala A, Goktepe M. Testing Metro style apps in Windows 8. Windows 8 app developer blog, http://blogs.msdn.com/b/windowsappdev/archive/2012/07/12/testing-metro-style-apps-in-windows-8.aspx, 2012.
[Sutton07] Sutton M, Greene A, Amini P. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional, 2007.
[TheTestEye11] The Test Eye. Software Quality Characteristics. http://thetesteye.com/posters/ TheTestEye_

SoftwareQualityCharacteristics.pdf, 2011.

[TheTestEye12] The Test Eye. 37 Sources for Test Ideas. http://thetesteye.com/posters/TheTestEye_SourcesForTestIdeas.pdf, 2012.

[Vijayaraghavan03] Vijayaraghavan G, Kaner C. Bug Taxonomies: Use Them to Generate Better Tests. http://www.testingeducation.org/BBST/testdesign/Vijay_bugtax.pdf, STAR EAST, 2003.

[Weinberg08] Weinberg G. *Perfect Software: And Other Illusions about Testing*. Dorset House Publishing Company, 2008.

[Weinberg09] Weinberg G. Perfect Software: And Other Illusions about Testing. http://secretsofconsulting.blogspot.com/2009/01/testability-and-reproducibility.html, 2009.

[Whittaker01] Whittaker J A. Bugs, Patterns, Automation: Thoughts on More Effective Testing. Talk in Microsoft, 2001.

[Whittaker02] Whittaker J A. *How to Break Software*. Addison-Wesley, 2012.

[Whittaker09] Whittaker J A. *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. Addison-Wesley Professional, 2009.

[Whittaker10] Whittaker J A. Turning Quality on its Head. https://docs.google.com/file/d/0B4fTBFGDn-QkMDE5OTk0ZmUtODQ4Ni00NTM3LTlkMGEtNjg1MTdiMzM5Nzg5/edit?pli=1, 5th Annual Google Test Automation Conference, Hyderabad, 2010.

[Whittaker11] Whittaker J A. The 10 Minute Test Plan. http://googletesting.blogspot.com/2011/09/10-minute-test-plan.html, 2011.

[Whittaker12] Whittaker J A, Arbon J, Carollo J. *How Google Tests Software*. Addison-Wesley Professional, 2012.

[WikipediaChecklist12] Wikipedia. Checklist. http://en.wikipedia.org/wiki/Checklist, 2012.

[WikipediaDogfooding13] Wikipedia. Eating your own dog food. http://en.wikipedia.org/wiki/Dogfooding, 2013.

[WikipediaERM12] Wikipedia. Entity-Relationship model. http://en.wikipedia.org/wiki/EntityRelationship_Model, 2012.

[WikipediaGEPB12] Wikipedia. Georege E. P. Box. http://en.wikipedia.org/wiki/George_E._P._Box, 2012.

[WikipediaHeuristic12] Wikipeda. Heuristic. http://en.wikipedia.org/wiki/Heuristic, 2012.

[WikipediaMBT12] Wikipedia. Model-based testing. http://en.wikipedia.org/wiki/Model_based_testing, 2012.

[WikipediaMBTI13] Wikipedia. Myers-Briggs Type Indicator. http://zh.wikipedia.org/wiki/MBTI. 2013.

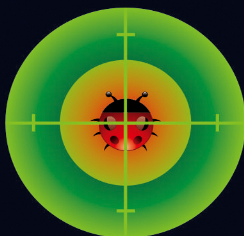
[WikipediaXUnit13] Wikipedia. xUnit. http://en.wikipedia.org/wiki/XUnit, 2013.

“在层出不穷的软件技术当中，软件测试一直扮演着幕后英雄的角色。没有测试，对发布者来说，就无法对质量有信心；对开发者来说，就有可能在修改代码的时候捅出了大娄子自己还不知道。小到代码中的一行assert，大到场景测试，本质上都是测试。专业测试人员固然需要系统学习测试，但在开发和测试的边界日趋模糊的今天，开发者学习测试也是很加分的。我在上大学的时候就认识史亮兄，他一直是厚积薄发的一个人，正是坐得了冷板凳，拿得出硬功夫。在微软做了一线测试工程师多年之后，再花上一整年的时间写作，才有这本著作，很值得一读。”

——**刘未鹏**（微软软件开发工程师，《暗时间》的作者）

“在所有软件测试的书中，这是我看过的最好的一本。看到有些章节的时候，都忍不住把里面的方法往手头的产品上靠。即便是粗略读一遍后获取的灵感和思路，都足够让我兴奋一周，更别提带来的实际价值了。软件测试是极其有趣的工作。作者不但精于此道，乐在其中，还能深入浅出地把各种思路、诀窍、心得甚至趣事跟大家共享，这将极大地推动人类文明的进程！”

——**熊力**（微软资深软件开发工程师，《Windows用户态程序高效排错》的作者）



图灵社区：iTuring.cn
热线：(010) 51095186 转 600

分类建议 计算机/软件测试

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-34584-4
定价：69.00元

看完了

如果您对本书内容有疑问，可发邮件至contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这里可以找到我们：

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks